The Dissertation Committee for Youngmoon Choi
certifies that this is the approved version of the following dissertation:

Parallel Prefix Adder Design

Committee:

Earl E. Swartzlander, Jr., Supervisor

Baxter F. Womack

Tony Ambler

Nur Touba

Chang Yong Kang

Parallel Prefix Adder Design

by

Youngmoon Choi, B.S., M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2004

Dedicated to my parents, my sister In-Young and my wife Min-A

for their endless love and support.

Acknowledgments

During the course of study at UT, Job's confessing in the Bible has been always challenging and promising words in my mind. *"But he knows the way that I take; when he has tested me, I will come forth as gold (Job 23:10)"*. Whenever I faced with hardships from study and new life experiences of being abroad, his words have always given me a faith and strength to move on. I would like to confess that Jesus was, is and will be a shepherd of my life and thank him for being a faithful friend as well as a shelter.

I would like to thank my supervisor, Prof. Earl E. Swartzlander, Jr., for his constant support. He has always guided me to the right directions with his profound academical insight. This is one of the best experiences of my life. I would like to thank Prof. Baxter F. Womack, Tony Ambler and Nur Touba for their excellent comments on my works. I am also very grateful to Eric Swanson and shall never forget the wonderful experience of working together at OASIS SiliconSystmes.

I am grateful to all the friends I've met here in Austin. First of all, I thank all the ASP Group members, especially, Chang-Yong Kang, Min Cha, Jaeki Yoo, Hyuk Park, Hempil Cho, and Jason Arbaugh who were wonderful companies to study together in the Lab. I was fortunate to have Sung-Wook Yu, Young-Jin Kim, Hyesoon Kim and Eunmi Park around when I first settled in Austin and would like to give my deep appreciation for their sincere help. I also thank Wonsik Chee,

Seok-Jin Lee, Jang-Won Lee, Sungkyu Song, Joonhyuk Kang, Kyoil Kim, Dong-Ho Kim, Hak-Soo Yu, Youngok Kim, Il-Soo Lee, Byeong-Kil Lee, Jaekwon kim and Hoosin Lee for their friendship and valuable time shared with me. I also thank for having all the activities with my ETC members, Seyoeung Choi, Jeahoon Jeong, Youngsang Kim, Jisun Park, Seung-Jun Baek, Haewoon Nam, Hyeon-Su Park and Changwoo Yang. It was such a pleasure to play tennis with them.

Finally, I am deeply indebted to my family in Korea, my parents, parents-in-law and sister, for their endless love and support. I always thank my wife, Min-A Kim, who showed me a deep trust and love on me all the time.

He replied,"Because your faith is too small. I tell you the truth, if your faith is as small as a mustered seed, you can say to this mountain, 'Move from here to there' and it will move. Nothing will be impossible for you." (Matthew 17:20)

Parallel Prefix Adder Design

Publication No. _____

Youngmoon Choi, Ph.D.
The University of Texas at Austin, 2004

Supervisor: Earl E. Swartzlander, Jr.

Adders are one of the critical elements in VLSI chips because of their variety of usages such as ALUs, floating point arithmetic units, memory addressing and program counting. For this reason, they have been studied for about half a century and a variety of adders have been invented. Among them, prefix adders are based on parallel prefix circuit theory which provides a solid theoretical basis for wide range of design trade-offs between delay, area and wiring complexity.

This dissertation first presents an algorithm for prefix computation under the condition of non-uniform input signal arrival. To obtain the algorithm, the structure of prefix circuits is analyzed and a generalized circuit structure that is composed of two parts, a full-product generation tree and sub-product generation trees, is proposed. For the full-product generation tree, a delay optimized design algorithm is proposed and its optimality is shown. The proposed algorithm is easy of implement and fast in run-time due to its greedy strategy and it ensures the minimum depth prefix circuit design with the Ladner-Fischer strategy.

This dissertation also presents a one-shot batch process that generates a wide range of designs for a group of parallel prefix adders. The prefix adders are represented by two two-dimensional matrixes and two vectors. This matrix representation makes it possible to compose two functions for gate sizing which calculate the delay and the total transistor width of the carry propagation graph of adders. After gate sizing, the critical path net list of the carry propagation graph is generated from the matrix representation for spice delay calculation. The process is illustrated by generating sets of delay and total transistor width pairs for 32-bit and 64-bit cases.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Adders are one of the critical elements in VLSI chips because of their variety of usages such as ALUs, floating point arithmetic units, memory addressing and program counting. For this reason, they have been studied for about half a century and a variety of adders have been invented [20][21]. Among them, Brent-Kung [3], Kogge-Stone [1], Ladner-Fisher [2], Han-Carlson [4] and Knowles [5] adders are based on parallel prefix circuit theory. The prefix circuit theory provides a solid theoretical basis for wide range of design trade-offs between delay, area and wiring complexity. This property is very attractive to designers because the importance of the optimal design is growing as the usage of VLSI chips are widen. For example, a VLSI chip for a huge industrial server surely has different design objectives compared to a chip for a small cellular phone and designers should optimize each chip according to its own design objectives.

Another trend of VLSI design is that time to market is getting shorter and the importance of automated design is increasing. Prefix adders also satisfy this need because they are easy to integrate into a CAD system due to their regularity.

In this dissertation, issues on prefix circuit and adder design are addressed. Traditionally, prefix circuits have been designed under the assumption that all the inputs arrive at the same time, but this uniform arrival time assumption is not true

for some applications such as final adders for fast parallel tree multipliers. Fast parallel tree multipliers [16–18] can be divided into three parts, one that forms the matrix of bit products, one that compresses the columns of the matrix into two rows each and one that sums the two rows. The timing of the output of the column compression part is not uniform but varies depending on the bit position. The first issue of this dissertation is how to use effectively the time redundancy of the non-uniform input profile under the framework of prefix circuit design.

The next issue of this dissertation is how to map a prefix circuit level design into a gate level implementation of the circuit. Because the number of possible prefix circuits increases exponentially as the number of inputs increases, an automated mapping process is needed to compare performances of a large number of mapped gate level implementations. Because the performance of the gate level implementation is highly dependant on gate sizing, the automated mapping process should be capable of gate sizing. The process should also be able to measure transistor area and delay for the optimal design of the gate level implementation.

1.1 Prefix Circuit

In this section, before explaining prefix adders, general definition for the prefix circuit is presented in advance.

Let $\circ$ be an arbitrary associative binary operation. A prefix circuit for $\circ$ is a combinational circuit which takes n inputs $x_1,\ x_2, \ldots, x_n$ and generates n outputs $x_1,\ x_1 \circ x_2,\ x_1 \circ x_2 \circ x_3, \ldots, x_1 \circ \cdots \circ x_n$ as shown in Figure 1.1.

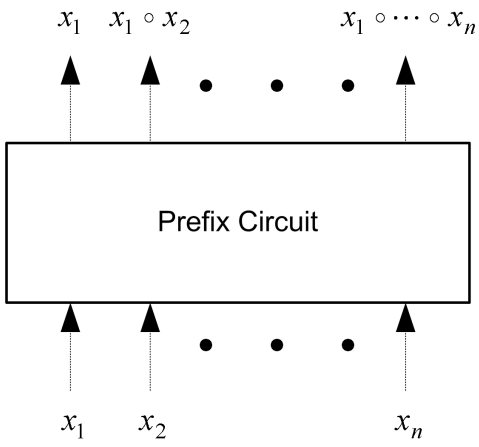

Figure 1.1: Function of prefix circuit.

1.2 Prefix Adder

To explain what the prefix adder is, it is compared with the full adder, the most famous adder ever. The biggest difference between two adders is that in the full adder, summation and carry calculation is done in the same one bit block but in the prefix adder, summation and carry calculation are separated from the bit block and all calculation is treated as a whole in the carry graph as shown in Figure 1.2. The carry graph uses the prefix circuit of Section 1.1 and this is the origin of the name, "Prefix Adder".

The following is the mathematical derivation of a prefix adder. Let $A = a_{n-1}a_{n-2}\ldots a_0$ and $B = b_{n-1}b_{n-2}\ldots b_0$ be n-bit binary numbers with sum $S = s_{n-1}s_{n-2}\ldots s_0$. The Least Significant Bit (LSB) is bit 0 and the Most Significant Bit (MSB) is bit n-1. If c_0 is carry-in signal to the LSB, the following equations can

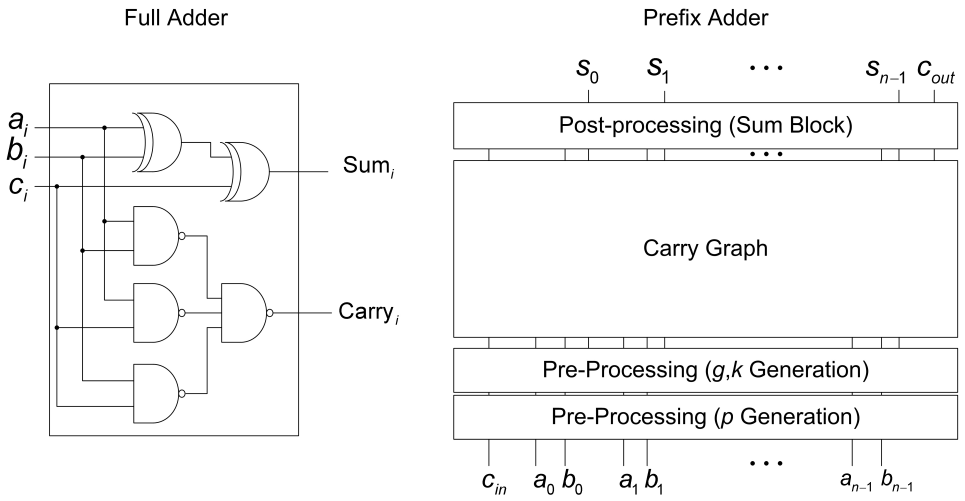

Figure 1.2: Full adder vs. prefix adder.

be used to compute the sum:

$$g_i = a_i b_i \tag{1.1}$$

$$k_i = \overline{a_i + b_i} \tag{1.2}$$

$$p_i = a_i \oplus b_i \tag{1.3}$$

$$c_{i+1} = g_i + \bar{k_i} c_i \tag{1.4}$$

$$s_i = p_i \oplus c_i \quad (\ i = 0,\ 1, \ldots, n-1) \tag{1.5}$$

where, g, k and p mean generate, kill and propagate, respectively.

The recurrence relation in Equation (1.4) can be applied repeatedly to obtain the following set of carry equations in terms of g_i, k_i and c_0.

$$c_{i+1} = g_i + (\sum_{j=0}^{i-1}(\prod_{k=j}^{i} \bar{k_k}) g_j) + (\prod_{k=0}^{i} \bar{k_k}) c_0 \quad (\ i = 0,\ 1, \ldots, n-1) \tag{1.6}$$

If binary operator $\circ$ is defined on ordered pair $(\bar{k}, g)$ by

$$(\bar{k_i}, g_i) \circ (\bar{k_j}, g_j) \stackrel{def}{=} (\bar{k_i}\bar{k_j},\ g_j + \bar{k_j}g_i) \tag{1.7}$$

then,

$$(1, c_0) \circ (\bar{k_0}, g_0) \circ \ldots \circ (\bar{k_i}, g_i) = (\bar{k_0}\bar{k_1} \ldots \bar{k_i}, c_{i+1}) \tag{1.8}$$

Now, the original addition problem is transformed into the prefix circuit problem presented in Figure 1.1.

The most important properties of the $\circ$ operator are its associativity and idempotency, i.e.,

$$\{(\bar{k_i}, g_i) \circ (\bar{k}_{i+1}, g_{i+1})\} \circ (\bar{k}_{i+2}, g_{i+2}) = (\bar{k_i}, g_i) \circ \{(\bar{k}_{i+1}, g_{i+1}) \circ (\bar{k}_{i+2}, g_{i+2})\} \tag{1.9}$$

and

$$\{(\bar{k_i}, g_i) \circ (\bar{k}_{i+1}, g_{i+1})\} \circ \{(\bar{k}_{i+1}, g_{i+1}) \circ (\bar{k}_{i+2}, g_{i+2})\} = (\bar{k_i}, g_i) \circ (\bar{k}_{i+1}, g_{i+1}) \circ (\bar{k}_{i+2}, g_{i+2}). \tag{1.10}$$

The associativity enables Equation (1.6) to be evaluated in parallel and the idempotency allows the sub-trees to overlap which provides some useful flexibility in the parallelization. All the prefix adders utilize those parallelism to calculate the result quickly.

1.3 Contribution and Dissertation Overview

In this dissertation, two innovations on prefix circuit and prefix adder problems are presented. The first is that a minimum depth prefix circuit design scheme

under the non-uniform input signal arrival condition is proposed using an algorithmic approach and its optimality is proved formally. The second is that a fast characterization process for the group of the Knowles adders [5] is proposed which can be used for selecting an adder during the early design phase. The process uses a matrix representation for the gate level design of the Knowles adders which is composed of two two-dimensional matrixes and two vectors.

The composition of this dissertation as follows.

In Chapter 2, basic prefix circuits and Knowles circuit are explained as preliminary information.

Chapter 3 examines the design of a hybrid prefix adder for the final adder for fast parallel multipliers, which use column compression reduction. The timing of the output of the column compression part is not uniform but varies depending on the bit position. The prefix graph scheme efficiently accommodates the non-uniform arrival times. Rules are presented for designing hybrid prefix adders under such conditions. The rules are applied to the final adder of 16×16 Dadda multiplier and they produces adders which are faster and less complex than previous works.

Based on promising results of Chapter 3, Chapter 4 examines a generalized prefix circuit problem which assumes non-uniform input signal arrival, unlike traditional prefix circuits which are designed under uniform signal arrival condition. First, the common structures of traditional prefix circuits are analyzed. The analysis results show that the structures of all the prefix circuits can be divided into two parts. Based on this observation, two step design process for the prefix circuits is proposed. For the first step, an optimization problem called 'fastest full-product

generation problem' is mathematically formulated. A solution of this problem is presented as an algorithmic form and the optimality of the algorithm is proved formally. Finally, using the proposed algorithm, prefix circuits are composed and they are compared with the traditional prefix circuits to show the usefulness of the proposed design.

In Chapter 5, a one-shot process analyzing characteristics of the complete set of Knowles prefix adders [5] is proposed. First, a matrix representation for the gate level design of the Knowles adders is proposed which is composed of two two-dimensional matrixes and two vectors. Second, delay and transistor width calculation functions for gate sizing are constructed from the matrixes. Because the matrix representation contains enough information to construct its original adder, it is also possible to generate spice net list from the matrixes. These produce a one-shot batch process generating the complete set of characteristic curves of the adders in a group.

Chapter 6 concludes this dissertation.

Chapter 2

Parallel Prefix Circuits

This chapter presents preliminary information for the rest of this dissertation. First, basic prefix circuits are summarized and next, hybrid forms, Han-Carlson and Knowles circuits are explained.

2.1 Basic Prefix Circuits

In Figure 2.1, three basic types of 8-bit prefix circuits (Kogge-Stone [1], Ladner-Fischer [2] and Brent-Kung [3]) are shown. The major differences of these

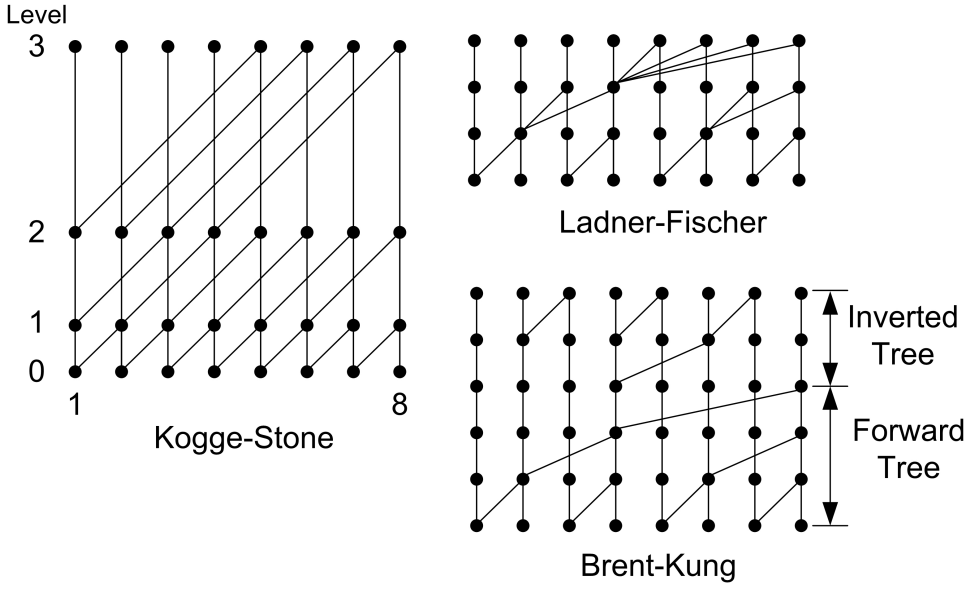

Figure 2.1: Basic prefix circuits.

prefix graphs are their paralleling strategies. Kogge-Stone and Ladner-Fischer obtain maximum parallelism by unlimited parallel connection and fan-out, respectively. On the contrary, Brent-Kung limits the parallelism and has more stages to reduce the complexity at the expense of speed.

2.2 Hybrid Circuits

Han-Carlson [4] presented a hybrid prefix circuit which uses a Brent-Kung prefix circuit (slow, but small) in the top and bottom rows, and a Kogge-Stone prefix circuit (fast, but large) in the center rows to optimize the total area and time for the circuit, as shown in Figure 2.2.

Knowles [5] presented complete classes of regular fan-out prefix circuits which are bounded at the extremes by the Ladner-Fischer and Kogge-Stone prefix circuits.

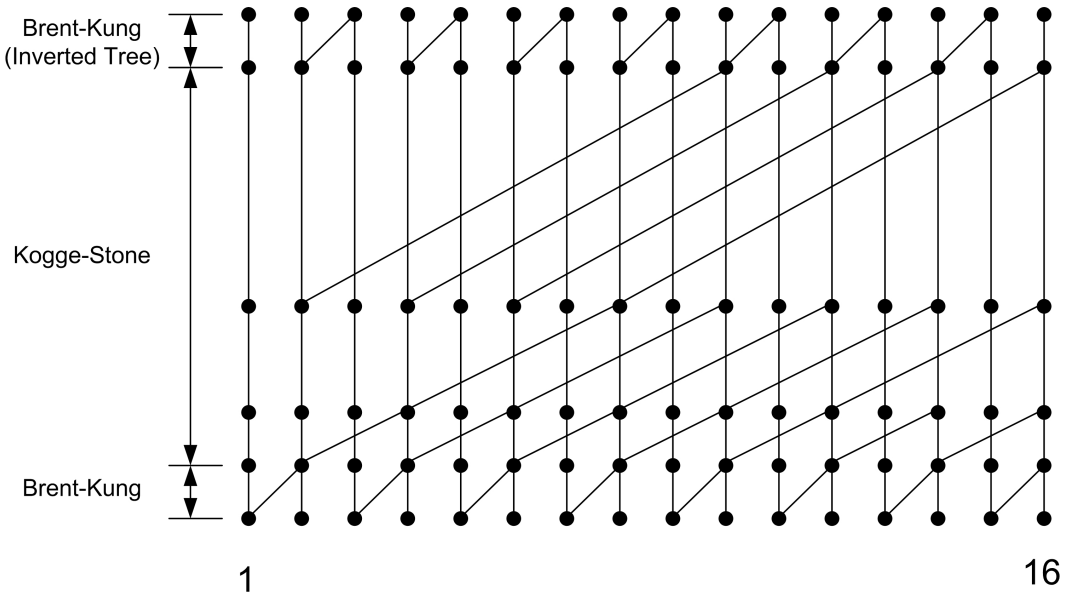

Figure 2.2: 16-bit Han-Carlson circuit.

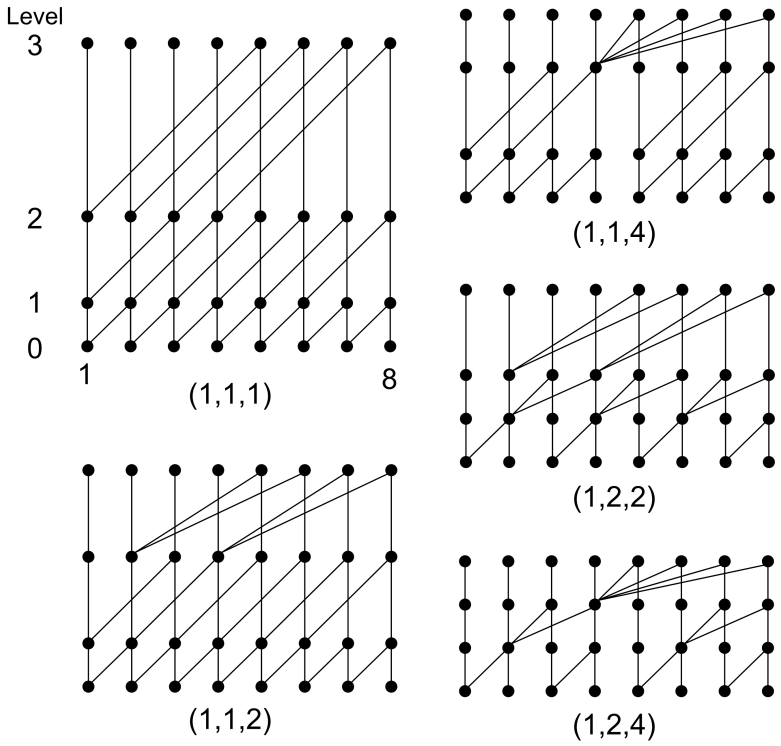

Figure 2.3: 8-bit Knowles circuits.

Knowles circuits of 8-bit width are presented in Figure 2.3. In the figure, (1,1,1) is the Kogge-Stone circuit and (1,2,4) is the Ladner-Fisher circuit.

The following rules are used for the construction of Knowles adders:

- Lateral wires at the j^{th} level span 2^j bits. $(0 \leq level \leq log_2 n)$

- The lateral fan-out at the j^{th} level is a power of 2 between 1 and 2^j inclusive.

- The lateral fan-out at the j^{th} level cannot exceed that at the $(j+1)^{th}$ level.

Because of these construction rules, Knowles circuits are very regular and the numbers of circuits can be counted according to operand widths as shown in

Table 2.1. The vector under each figure in Figure 2.3 represents the lateral fan-out of each level. From now on, the vector is called fan-out vector and expressed as follows:

$$fo = (fo(0), \ldots, fo(log_2 n - 1)),\ where\ n = operand\ width. \tag{2.1}$$

Each fan-out vector corresponds to a circuit in the group as shown in Figure 2.3.

Table 2.1: Number of Knowles Circuits.

Operand Width (bits)	Number of Knowles Circuits
4	2
8	5
16	14
32	42
64	132
128	429
256	1430

Chapter 3

Design of a Hybrid Prefix Adder for Fast Multipliers

This chapter examines the design of a hybrid prefix adder under the condition of non-uniform input signal arrival. This is encountered in the final adder for fast parallel multipliers which use column compression reduction. The prefix graph scheme efficiently accommodates the non-uniform arrival times. Rules are presented for designing hybrid prefix adders under such conditions. This rule produces adders which are faster and less complex than previous works.

3.1 Introduction

Fast parallel tree multipliers [16–18] can be divided into three parts, one that forms the matrix of bit products, one that compresses the columns of the matrix into two rows each and one that sums the two rows. The timing of the output of the column compression part is not uniform but varies depending on the bit position. It is possible to use this profile to develop a more efficient carry propagation adder than a normal adder that is designed assuming uniform input arrival times.

A typical profile of the arrival time of inputs to the carry propagating adder of a fast multiplier has three regions. At the least significant end, successive bits are available later than their less significant neighbors. In the middle the bits arrive approximately at the same time, and at the most significant end, successive bits

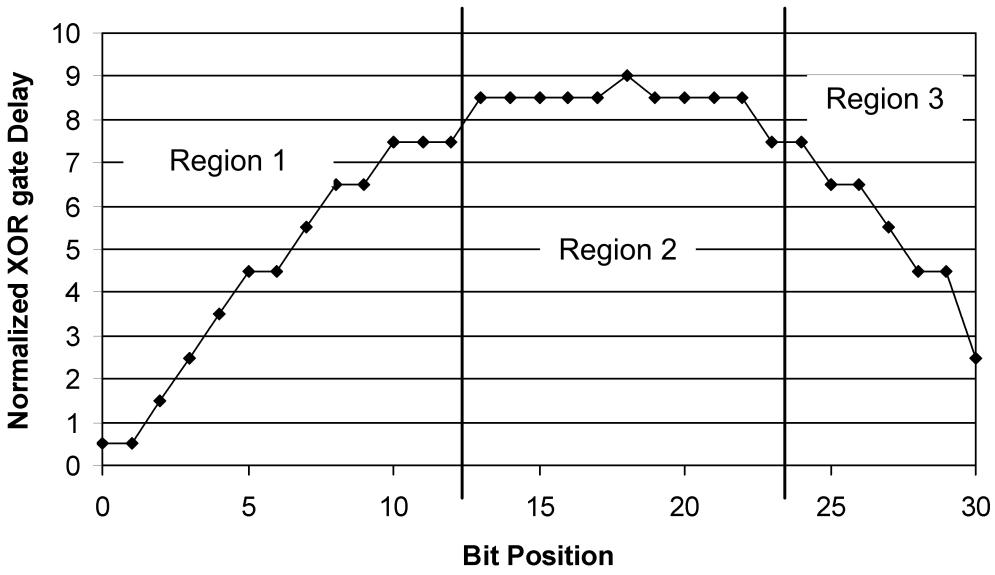

Figure 3.1: Signal arrival profile of a 16×16 Dadda multiplier and its regions.

are available earlier than their less significant neighbors [18, 19]. This is because the columns of bit products of a multiplier have the most rows in the center of the matrix. For example, Figure 3.1 depicts the arrival times of the bits to the final adder of a 16×16 Dadda multiplier [16]. The figure clearly shows the there are three regions, the positive slope region for the least significant bits, the flat region for the middle bits and the negative slope region for the most significant bits.

The fast multipliers described in [18, 19] use a ripple carry adder or a variable block size carry skip adder in the positive slope region (region 1) and a carry select or conditional sum adder in the flat and negative slope regions (region 2, 3). It is difficult to find the optimal break point between region 1 and 2, because the profile is

not a straight line as shown in Figure 3.1. Moreover, the overall size of the proposed adder is quite large, because both carry select and conditional sum adders need a lot of gates and they use the scheme over the large span of regions 2 and 3.

In this chapter, hybrid prefix adders are used to overcome those problems. As explained in Section 2.2, Han-Carlson [4] presented a hybrid prefix adder which uses a Brent-Kung prefix circuit (slow, but small) in the top and bottom rows, and a Kogge-Stone prefix circuit (fast, but large) in the center rows to optimize the total area and time for the adder, as shown in Figure 2.2. A similar idea is used to exploit the non-uniform arrival of the bits to the final adder. In the positive slope region (the least significant bits), the Brent-Kung prefix graph adder is used, because there is enough time for the calculation. In the flat region (middle bits), the Kogge-Stone Prefix graph adder is used for the fast calculation. Since the negative slope region (most significant bits) also has some time redundancy, the Brent-Kung scheme can be used in this region. Because the inputs come earlier than LSB carries, a kind of carry select scheme is attractive in this region. Thus, a modified carry select Brent-Kung adder is used.

The rest of this chapter is organized as follows. In Section 3.2, suitable hybrid prefix schemes according to the input patterns are examined, and in Section 3.3, the proposed schemes are applied to the final adder of 16×16 Dadda multiplier. Section 3.4 concludes this chapter.

3.2 Hybrid Prefix Adder Design

3.2.1 Leaf cells of Han-Carlson Adder

There are 4 different types of leaf cells in the standard Han-Carlson adder: kggen, black, white, and sum.

i) kggen cell(k, g generate cell)

$$k_i = \overline{a_i + b_i}, \quad \bar{g_i} = \overline{a_i b_i}$$

ii) black cell (k, g propagation cell)

bn (negative input)

$$g_j = \overline{(\bar{g_j}(k_j + \bar{g_i}))}, \quad \bar{k_j} = \overline{(k_i + k_j)}, \quad where\ j > i$$

bp (positive input)

$$\bar{g_j} = \overline{(g_j + \bar{k_j} g_i)}, \quad k_j = \overline{(\bar{k_i}\bar{k_j})}, \quad where\ j > i$$

iii) white cell (invert cell)

$$\{\bar{g_j} = g_j,\ k_j = \bar{k_j}\} \ \ or \ \ \{g_j = \bar{g_j},\ \bar{k_j} = k_j\}$$

iv) Sum cell

$$s_i = \overline{(c_{i+1} + k_i \bar{c_i})(\bar{g_i} + \bar{c_i})}$$

3.2.2 Positive Slope and Flat Region

If the left input of the operator $\circ$ of the black cell (k,g propagation cell) is the carry, then the output of this cell is also a carry. Using this property, a prefix graph similar to a ripple carry adder can be made. For this, one more k,g propagation cell is introduced:

bn_bar

$$\bar{g_j} = (\bar{g_j}(k_j + \bar{g_i})), \quad k_j = (k_i + k_j), \quad where\ j > i.$$

Because the signs of inputs and outputs of the black cells are opposite, they can be made with AOI (AND OR INVERTER) or OAI (OR AND INVERTER) gates, so they have about half of the delay of an XOR gate delay. On the other hand, since the signs of inputs and outputs of bn_bar cell are the same, they should be made with two distinct gates, and the delay is similar to that of an XOR gate. In Figure 3.2, mixed ripple carry and Brent-Kung prefix graph schemes are depicted for various input patterns.

Once the propagation of the carrys meets the latest arrival input groups, the Kogge-Stone Prefix graph is used to decrease the overall maximum delay. If the latest arrival input groups may have some irregular patterns, they can be effectively flattened with small normal Brent-Kung and/or ripple carry Brent-Kung trees.

3.2.3 Negative Slope Region

For this region, the carry select scheme is used because inputs of this region arrive earlier than inputs of flat region. To propagate two pairs of k and g at the

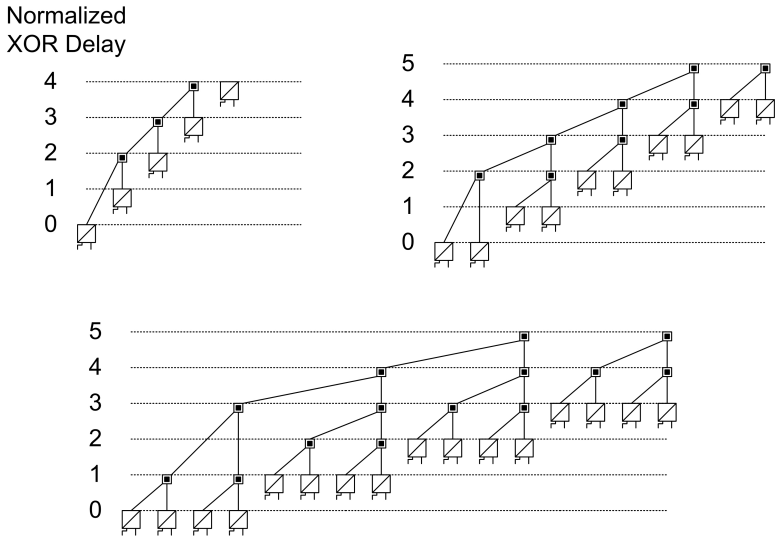

Figure 3.2: Mixed ripple carry and Brent-Kung prefix graph for various input patterns (Inverted trees are not shown in this figure, for simplicity).

same time, another type of a cell is employed, called a 'double k, g propagation cell'. It has two pairs of left inputs $((\bar{k_i^1},\ g_i^1),\ (\bar{k_i^2},\ g_i^2))$, one pair of center inputs $(\bar{k_j},\ g_j)$ and two pairs of k, g outputs, i.e.:

double k, g propagation cell

$$(\bar{k_i^1}, g_i^1) \circ (\bar{k_j}, g_j) = (\bar{k_i^1}\bar{k_j},\ g_j + \bar{k_j} g_i^1),$$
$$(\bar{k_i^2}, g_i^2) \circ (\bar{k_j}, g_j) = (\bar{k_i^2}\bar{k_j},\ g_j + \bar{k_j} g_i^2). \tag{3.1}$$

Other double cells are no more than a pair of single cells of that type. In Figure 3.3, two types of 7-bit carry select prefix graph adders are presented, one is designed by the Kogge-Stone scheme and the other one is designed by the Brent-Kung scheme. The gate complexities of both adders are almost the same, but the Brent-Kung scheme has less routing because its connections are more localized. One

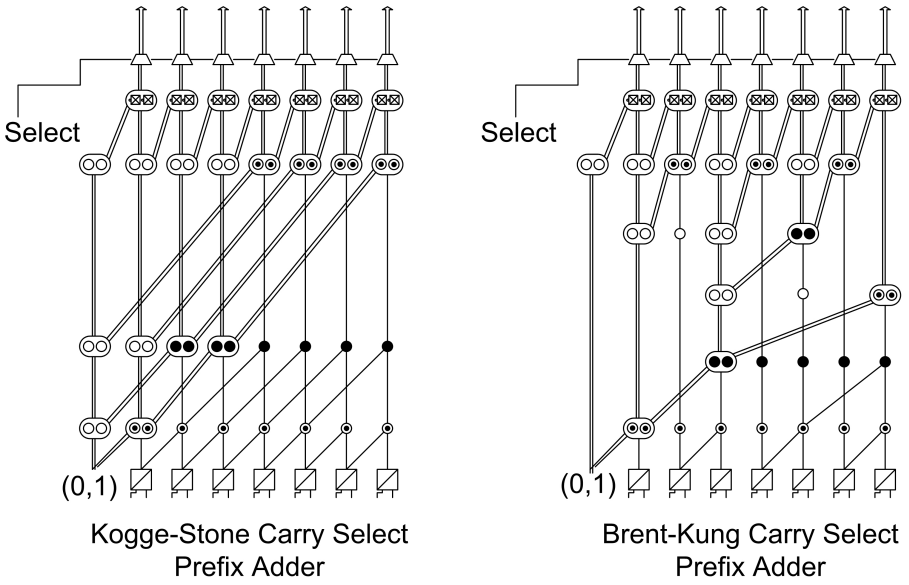

Figure 3.3: Carry select prefix adders.

of the interesting properties of the carry select prefix adders is that they need double cells at the upper parts only.

Between the two types of adders, the Brent-Kung carry select adder is used because the Brent-Kung scheme is more suitable for exploiting irregular patterns of input signals and an area efficient adder is more important here than a fast adder for the most significant bit part where time redundancy exists. Because this region requires a lot of area, our goal is to minimize this region as long as the critical path delay of the Kogge-Stone adder is not increased. Considering this fact and final mux delay, the input bit, which arrives earlier by more than two normalized XOR gate delay than the latest input group, can be a start bit of this region.

Table 3.1 and Figure 3.4 summarize the equations and symbols of all the leaf cells.

Table 3.1: Equation of Leaf cells (*: Leaf cells from Han-Carlson Adder).

Cell Name		Equation
kg Generate Cell*		$k_i = \overline{a_i + b_i}, \quad \bar{g}_i = \overline{a_i b_i}$
Black Cell ($j > i$)	bp*	$\bar{g}_j = \overline{(g_j + \bar{k}_j g_i)}, \quad k_j = \overline{(\bar{k}_i \bar{k}_j)}$
	bn*	$g_j = \overline{(\bar{g}_j (k_j + \bar{g}_i))}, \quad \bar{k}_j = \overline{(k_i + k_j)}$
	bn_bar	$\bar{g}_j = (\bar{g}_j (k_j + \bar{g}_i)), \quad k_j = (k_i + k_j)$
White Cell*		$\{\bar{g}_j = g_j,\ k_j = \bar{k}_j\} \ \ or \ \ \{g_j = \bar{g}_j,\ \bar{k}_j = k_j\}$
Sum Cell	ci_bar,c_i+1*	$s_i = \overline{(c_{i+1} + k_i \bar{c}_i)(\bar{g}_i + \bar{c}_i)}$
	ci,c_i+1_bar*	$s_i = \overline{(\bar{c}_{i+1} + k_i c_i)(\bar{g}_i + c_i)}$
	ci_bar, c_i+1_bar	$s_i = (\bar{c}_{i+1} + k_i \bar{c}_i)(\bar{g}_i + \bar{c}_i)$
Double Cell	D_bp	$\bar{g}_j{}^1 = \overline{(g_j + \bar{k}_j g_i^1)}, \quad k_j{}^1 = \overline{(\bar{k}_i{}^1 \bar{k}_j)}$ $\bar{g}_j{}^2 = \overline{(g_j + \bar{k}_j g_i^2)}, \quad k_j{}^2 = \overline{(\bar{k}_i{}^2 \bar{k}_j)}$
	D_bn	$g_j{}^1 = \overline{(\bar{g}_j (k_j + \bar{g}_i{}^1))}, \quad \bar{k}_j{}^1 = \overline{(k_i^1 + k_j)}$ $g_j{}^2 = \overline{(\bar{g}_j (k_j + \bar{g}_i{}^2))}, \quad \bar{k}_j{}^2 = \overline{(k_i^2 + k_j)}$
	D_ci,c_i+1_bar	$s_i{}^1 = \overline{(\bar{c}_{i+1}^1 + k_i^1 c_i{}^1)(\bar{g}_i{}^1 + c_i{}^1)}$ $s_i{}^2 = \overline{(\bar{c}_{i+1}^2 + k_i^2 c_i{}^2)(\bar{g}_i{}^2 + c_i{}^2)}$
	D_white	$(\bar{g}_j{}^1 = g_j{}^1,\ k_j{}^1 = \bar{k}_j{}^1), (\bar{g}_j{}^2 = g_j{}^2,\ k_j{}^2 = \bar{k}_j{}^2)$ or $(g_j{}^1 = \bar{g}_j{}^1,\ \bar{k}_j{}^1 = k_j{}^1), (g_j{}^2 = \bar{g}_j{}^2,\ \bar{k}_j{}^2 = k_j{}^2)$

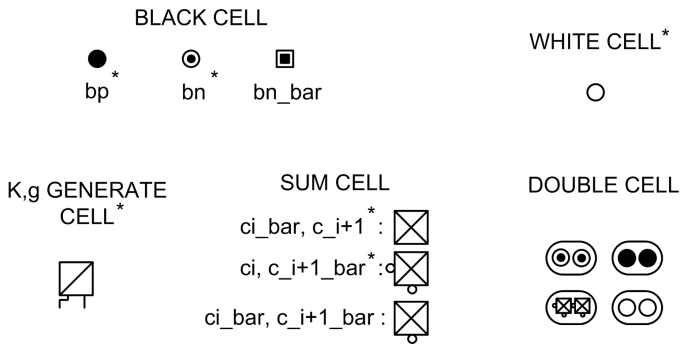

Figure 3.4: Notations of leaf cells.

3.3 Results

To obtain an input profile for the adders, first, the delay pattern of the column compression part of a 16×16 Dadda multiplier was analyzed assuming the characteristics of Table 3.2 for full adders and half adders.

Table 3.2: Adder Delay Models [Normalized XOR Delay].

	Half Adder	Full Adder
Input to Sum	1	2
Input to Carry	0.5	1
Carry to Sum	N/A	1

Based on the design scheme shown in Figure 3.5, the arrival time to the final adder which is shown in Figure 3.6 is obtained. Because this analysis uses the XOR gate delay model of Table 3.2 and connects the slow outputs to fast inputs to minimize the delay, the result shows a more refined delay profile than that of the normal Dadda multiplier using the unit adder delay model. The delay for partial product generation was assumed to be of 0.5 XOR gate delay since two input AND

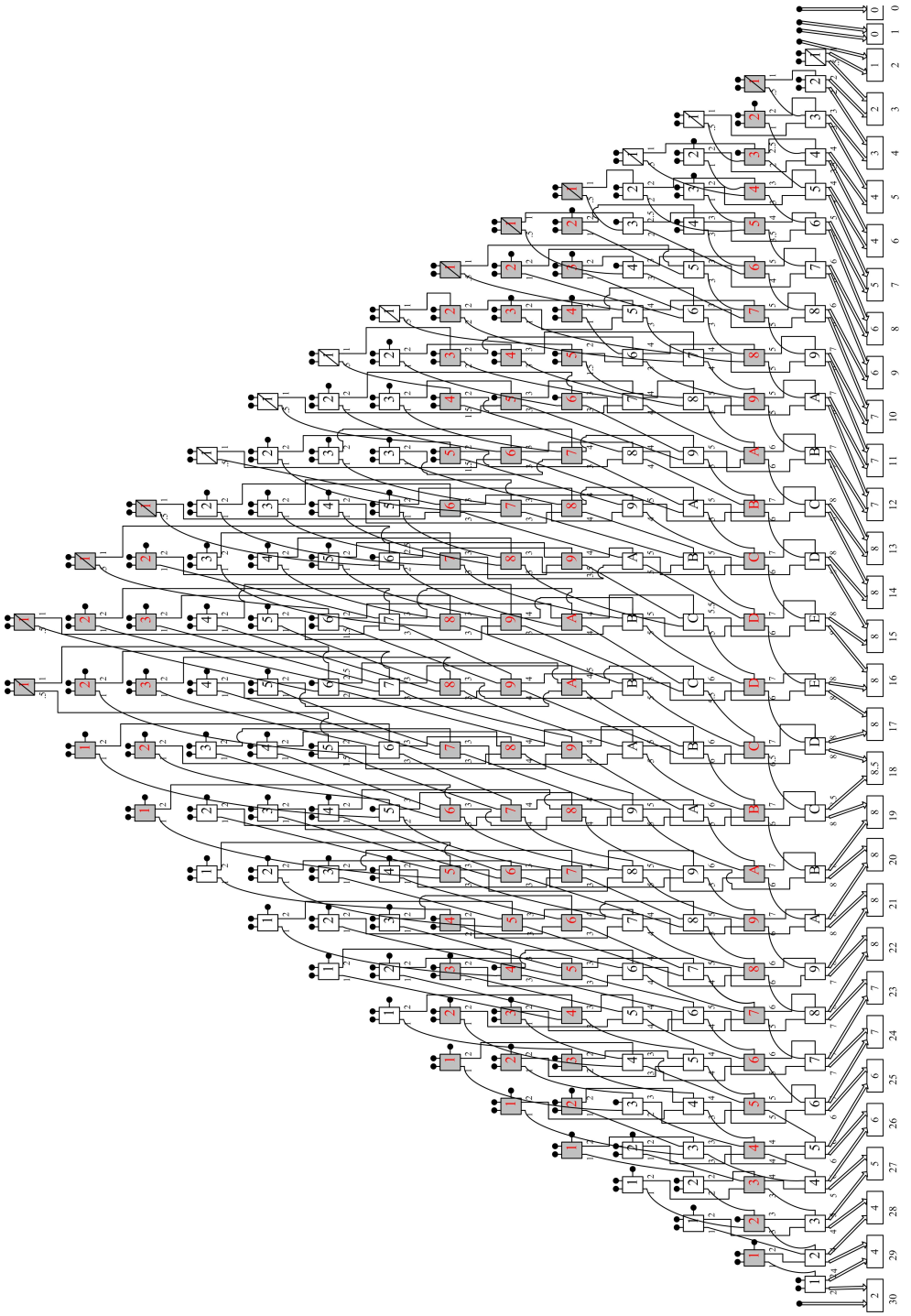

Figure 3.5: Delay analysis diagram for 16×16 Dadda multiplier.

gates were used. The analysis result of the maximum delay of column compression part is 8.5 XOR gate delay, so the overall maximum delay of the multiplier is 9 XOR gate delay.

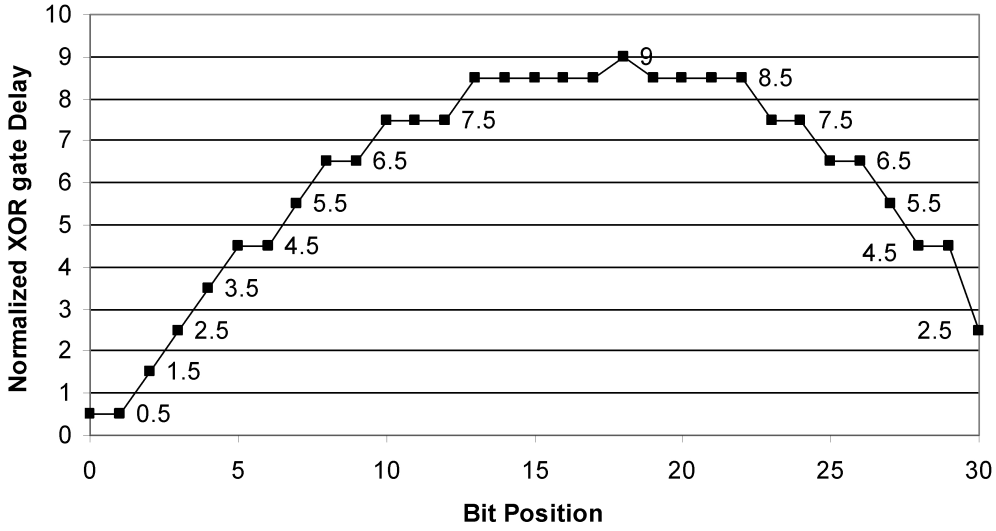

Figure 3.6: Signal arrival profile of the 16×16 Dadda multiplier.

Figure 3.7 is a time-prefix graph diagram of the proposed design, and Figure 3.8 is a rearrangement of Figure 3.7 for complexity comparison purposes.

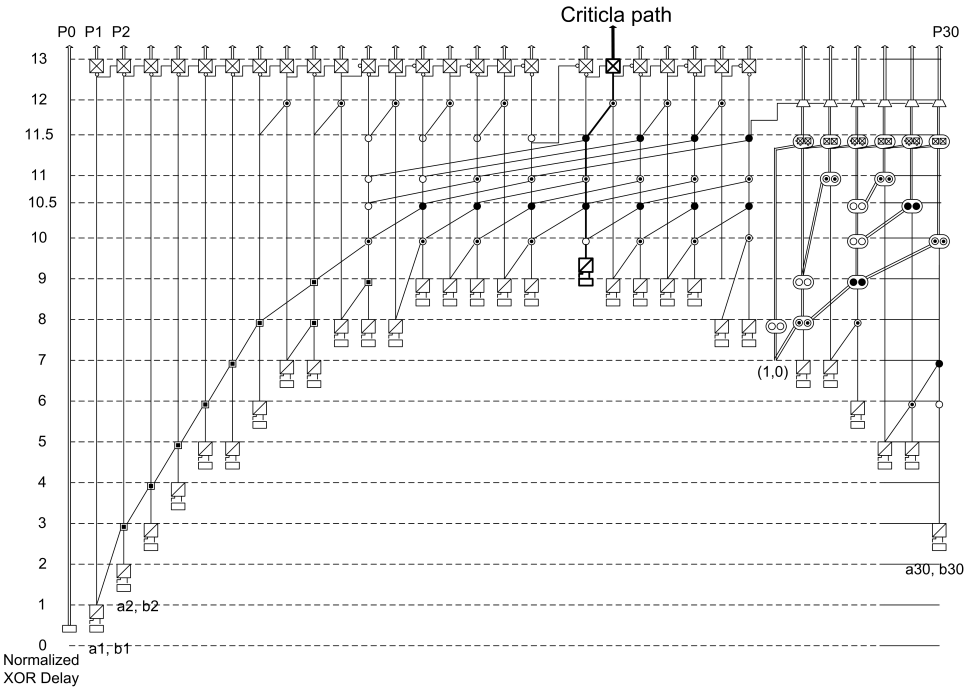

Figure 3.7: Critical path analysis of the proposed design.

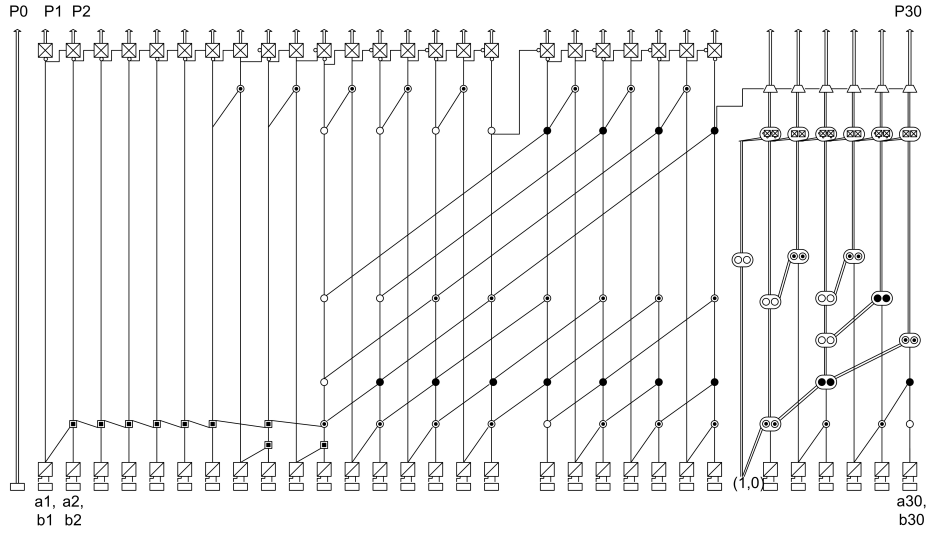

Figure 3.8: Rearrangement of Figure 3.7 for complexity comparison.

The proposed design is compared with the normal Han-Carlson adder shown in Figure 3.9, and the hybrid adder [18] shown in Figure 3.10.

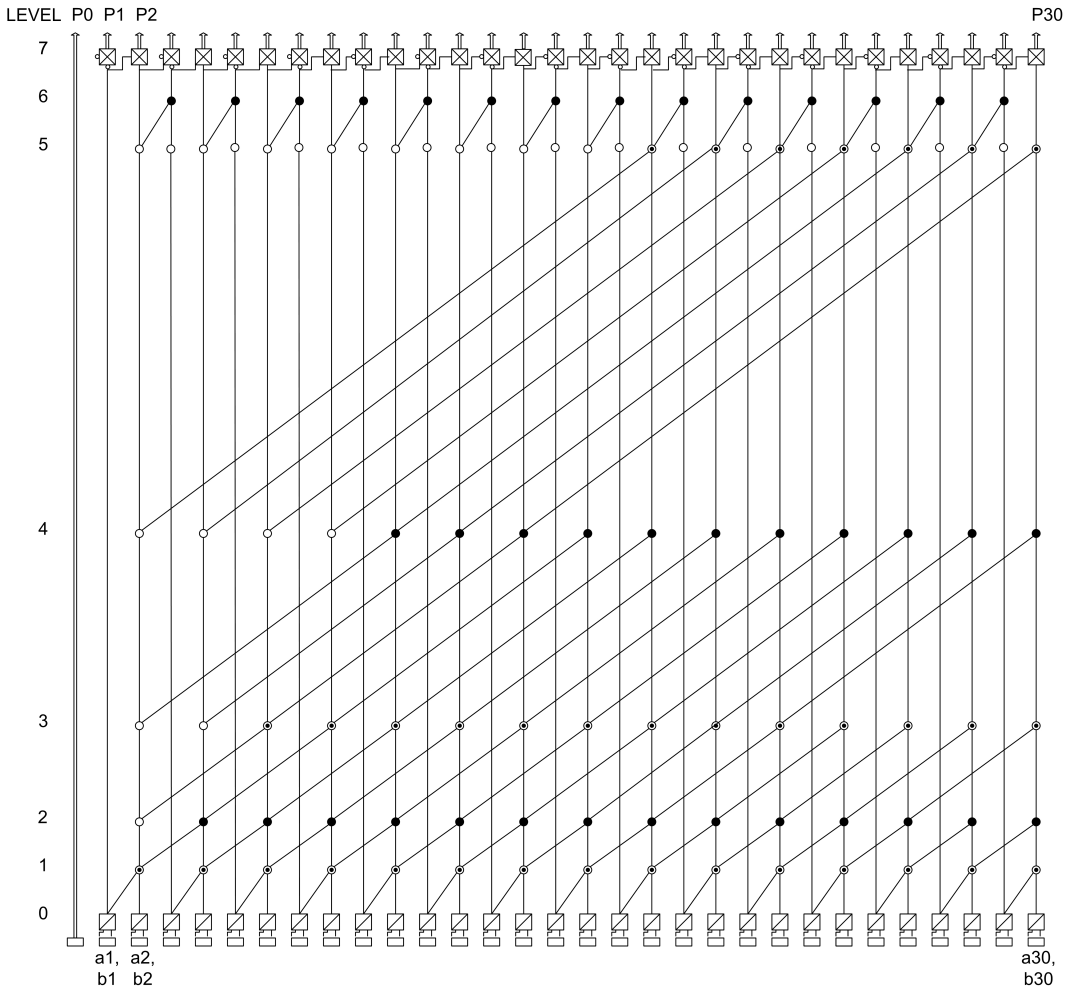

Figure 3.9: 30-bit Han-Carlson adder.

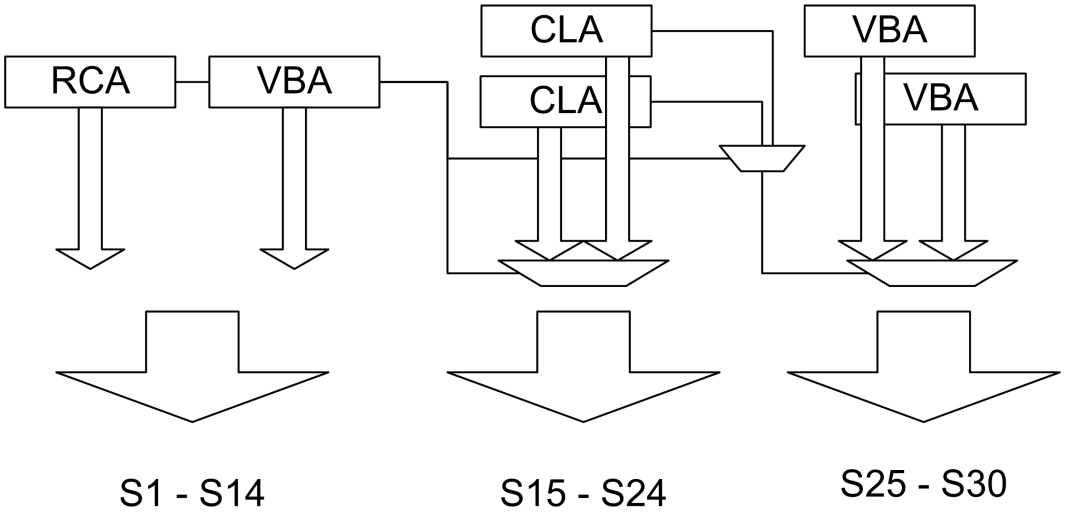

Figure 3.10: 30 bit hybrid adder proposed in [18] (region 1: 1-14, region 2: 15-24, region 3: 25-30) (RCA: Ripple Carry Adder, CLA: Carry Lookahead Adder, VBA: Variable Block size Adder).

Tables 3.3 and 3.4 summarize the results. In Table 3.3, the proposed design and Han-Carlson design have almost the same complexity and either is less complex than the hybrid design of [18], but this result does not include the routing complexity. As shown in Figure 3.8, the proposed design has much shorter routing lines than the Han-Carlson adder of Figure 3.9. Thus the overall complexity of the proposed design, which includes routing lines, is expected to be less than Han-Carlson design. As shown in the delay comparison results of Table 3.4, the proposed design is the fastest of the three adders. It is about 18% faster than the other adders.

Table 3.3: Complexity Comparison [Normalized Full Adder Delay].

	Cell Type	No.	Normalize Factor	Complexity
Proposed Design	White Cell	17	1/9	2
	Black Cell	47	1/3	15
	Sum Cell	36	1/2	18
	kg Generate Cell	30	1/4	7
	bn_bar	9	1/2	5
	2:1 MUX	6	1/6	1
	Total (Remarks)	48		
Han -Carlson [4]	White Cell	29	1/9	3
	Black Cell	74	1/3	24
	Sum Cell	30	1/2	15
	kg Generate Cell	30	1/4	7
	Total (Remarks)	49 (require large routing)		
Hybrid [18]	FA	46	1	46
	CLA-4b	4	2	8
	CLA-2,3b	4	1	4
	2:1 MUX	16	1/6	2
	Total (Remarks)	60		

Table 3.4: Delay Comparison [Normalized XOR Delay].

Proposed Design					
Critical Path	1×(2 Input)	1×Inv.	4×AOI	Sum	
Delay	0.5	0.4	$2.8 = 4 \times 0.7$	1	
Total Delay (Remarks)	4.7				
Han-Carlson [4]					
Critical Path	1×(2 Input)	6×AOI	Sum		
Delay	0.5	$4.2 = 6 \times 0.7$	1		
Total Delay (Remarks)	5.7 (require large routing)				
Hybrid [18]					
Critical Path	1×(2 Input)	2×CLA-4b	1×CLA-3b	1×XOR	MUX
Delay	0.5	$2.4 = 2 \times 1.2$	1	1	0.8
Total Delay (Remarks)	5.7 (assume critical path lies in region 2)				

3.4 Conclusion

As shown in the example of Section 3.3, the proposed design has advantages in both speed and complexity. The main merit of the proposed design is the reduction of the prefix graph complexity in the critical path. A normal 30 bit Han-Carlson adder has very long routing lines at the last level of the Kogge-Stone tree, but, by using the time difference of the inputs, the proposed design entirely eliminates the last level of the Kogge-Stone tree and more than half of the next level tree. Because of this reduction, the proposed design is expected to offer better performance than either of the previous designs.

Chapter 4

Prefix Circuit for Non-Uniform Input Arrival Times

Based on promising results of Chapter 3, this chapter examines a generalized prefix circuit problem which assumes non-uniform input signal arrival and an algorithm is proposed as a solution. To obtain the algorithm, the structure of prefix circuits is analyzed and a generalized circuit structure that is composed of two parts, a full-product generation tree and sub-product generation trees, is proposed. For the full-product generation tree, a delay optimized design algorithm is proposed and its optimality is shown. The proposed algorithm is easy of implement and fast in run-time due to its greedy strategy and it ensures the minimum depth prefix circuit design with the Ladner-Fischer strategy.

4.1 Introduction

For prefix computation, a variety of schemes have been presented under the assumption that all the inputs arrive at the same time. However, this uniform arrival time assumption is not true for some applications such as final adders for fast parallel multipliers as shown in Chapter 3. The non-uniform input arriving condition changes the prefix problem into a time optimization problem and traditional prefix schemes cannot guarantee minimum delay solutions. Like almost all such problems, the complexity of this problem increases exponentially as the number of inputs are

increased. Thus, this chapter presents a framework and an algorithm to solve the problem.

In this chapter, first, the common structures of traditional prefix circuits are analyzed. The analysis results show that the structures of all the prefix circuits can be divided into two parts. Based on this observation, two step design process for the prefix circuits is proposed. For the first step, an optimization problem called 'fastest full-product generation problem' is mathematically formulated. A solution of this problem is presented as an algorithmic form and the optimality of the algorithm is proved formally.

The rest of this chapter is organized as follows. In Section 4.2, a generalized structure of prefix circuit is proposed. In Section 4.3, Fastest Full-Product Generation Problem (FFP-Problem) is formulated and Fastest Full-Product Generation Algorithm (FFP-Algorithm) is proposed as a solution for the problem. The optimality of the algorithm is also proved in the section. In Section 4.4, prefix circuits are designed using the FFP-Algorithm. Section 4.5 concludes this chapter.

4.2 Generalized Structure of Prefix Circuit

As described in Section 1.1, if $\circ$ be an arbitrary associative binary operation, a prefix circuit for $\circ$ is a combinational circuit which takes n inputs $x_1,\ x_2, \ldots, x_n$ and generates n outputs $x_1,\ x_1 \circ x_2,\ x_1 \circ x_2 \circ x_3, \ldots, x_1 \circ \cdots \circ x_n$. In this dissertation, $x_1 \circ \cdots \circ x_n$ is called the 'full-product' and the rest of products, $x_1 \circ \cdots \circ x_k$ $(1 \leq k \leq n-1)$ are called 'sub-products'.

Because the purpose of the prefix circuit is to calculate all the products of

their inputs efficiently, the full-product $x_1 \circ \cdots \circ x_n$ should be implemented in a part of the prefix circuit. There are numerous ways to implement it. Obviously, the slowest way is to connect all nodes in series like a ripple carry adder and the fastest way is to make a radix-2 tree. Even though Ladner-Fischer, Kogge-Stone and Brent-Kung in Figure 2.1 are different from each other in design strategies, they have the same full-product generation tree, the fastest 8-bit radix-2 binary tree as shown in Figure 4.1. The important property of the tree for $x_1 \circ \cdots \circ x_n$ is that it sets the lower delay bound of the overall circuit and therefore the three basic prefix adders are using the fastest tree for it.

Based on observations above, a generalized structure for prefix circuits is proposed in Figure 4.2. The structure is composed of overlapping trees, the Full-Product Generation Tree (FP-Tree) and Sub-Product Generation Trees (SP-Trees). This generalized structure implies that the design process of all the prefix adders can be divided into two steps, the first step for the FP-Tree and the second step for SP-Trees.

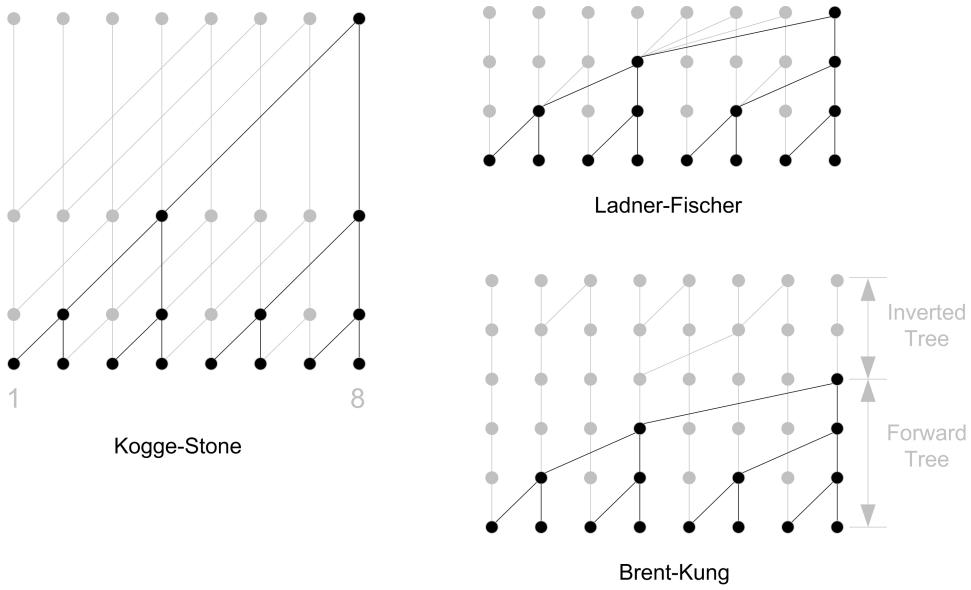

Figure 4.1: Full-Product generation trees of basic prefix circuits.

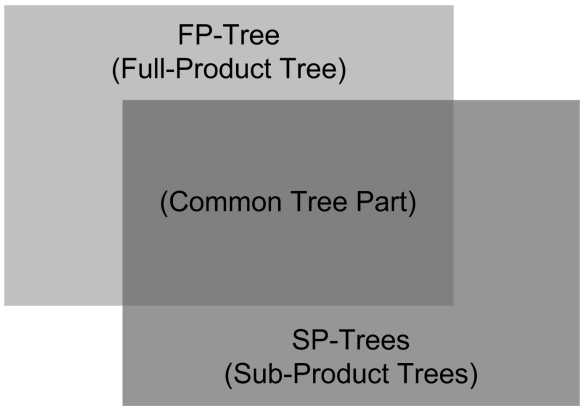

Figure 4.2: Generalized structure of prefix graph.

4.3 Fastest Full-Product Generation Tree

All the previous prefix adders in Chapter 2 are designed under the assumption that all the bits of the inputs arrive at the same time to the adders. The prefix adder design problem, however, can be generalized by removing the uniform input arrival constraint.

As noted in Section 4.2, the prefix adder design procedure can be divided into two steps. The first step is to design the FP-Tree and the second step is to design the SP-Trees. In this section, the FP-Tree design procedure is examined in detail.

4.3.1 Operation Order Profile

Let $X = (x_1,\ x_2, \ldots, x_n)$ be an input vector. All the orders of $\circ$ operations produce the same value for the full-product $x_1 \circ \cdots \circ x_n$ because the $\circ$ operator is associative. However, different orders of $\circ$ operations make different connection patterns for vertexes and result in structurally different FP-Trees. To assign an order to the $\circ$ operator, it is changed into $\overset{\alpha_i}{\circ}$, where each α_i has distinct integer value from 1 to n-1 that corresponds to the order of the i^{th} operation among the n-1 operations. The vector which is composed of α_i's is represented by $\theta = (\alpha_1,\ \alpha_2, \ldots, \alpha_{n-1})$ and called 'operation order profile'. Figure 4.3 depicts the 4 input case.

Because n-1 $\circ$ operations are needed to calculate full-product $x_1 \circ \cdots \circ x_n$, there exist $(n-1)!$ orders of $\circ$ operations for the n input case. However, the total number of structurally different FP-Trees may be less than $(n-1)!$ because different θ's may construct the same FP-Tree.

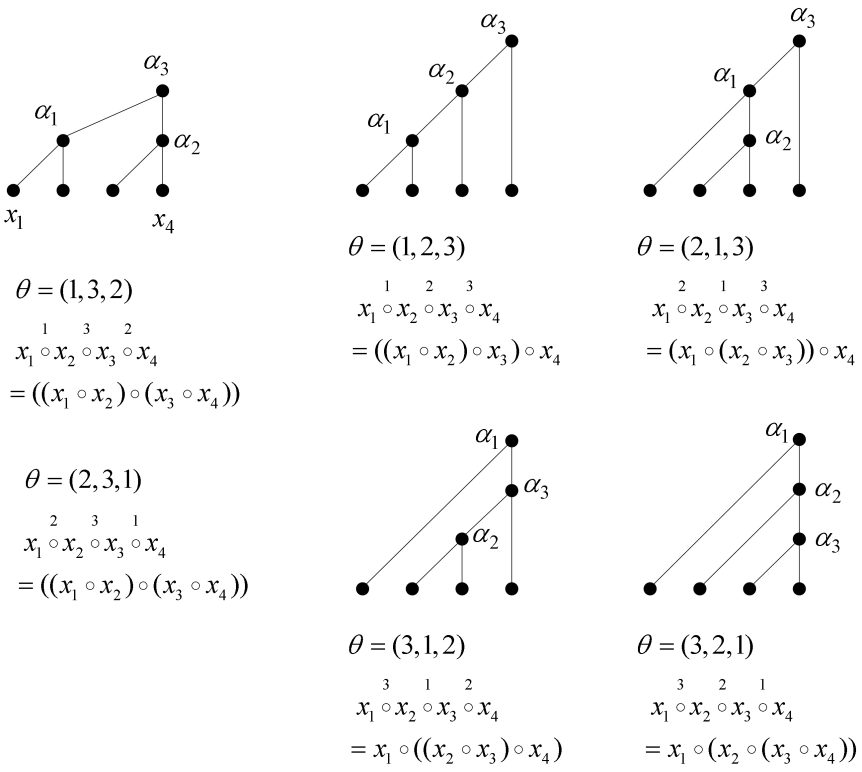

Figure 4.3: Complete set of FP-Trees and operation order profiles (number of input=4).

To avoid confusion between the computational equation for a tree and tree itself, $x_1 \circ \cdots \circ x_n$ represents the computational equation for a full-product and, on the other hand, $x_1 \overset{\alpha_1}{\circ} x_2 \overset{\alpha_2}{\circ} \cdots \overset{\alpha_{n-1}}{\circ} x_n$ represents the FP-Tree which is constructed by an operation order profile $\theta = (\alpha_1,\ \alpha_2, \ldots, \alpha_{n-1})$, from now on.

Definition 4.3.1. *$Tree_{FP}(X, \theta)$, where $X = (x_1, \ldots, x_n)$ and $\theta = (\alpha_1, \ldots, \alpha_{n-1})$.*

$Tree_{FP}(X, \theta) \overset{def}{=} x_1 \overset{\alpha_1}{\circ} x_2 \overset{\alpha_2}{\circ} \cdots \overset{\alpha_{n-1}}{\circ} x_n \overset{def}{=}$FP-Tree of X which is constructed by θ .

The set of all the operation order profiles of X is defined by $\Theta(X)$ and therefore $\Theta(X)$ is equivalent to the set of all the possible FP-Trees of the input vector X.

After a FP-Tree is constructed with $\theta = (\alpha_1, \ldots, \alpha_{n-1})$, each $\overset{\alpha_i}{\circ}$ corresponds to a vertex in the FP-Tree as shown in Figure 4.3. Let α_{max}=$max\{\alpha_1, \ldots, \alpha_{n-1}\}$, then $\overset{\alpha_{max}}{\circ}$corresponds to the top vertex of the FP-Tree. Using an analogy of this property, part of the FP-Tree equation can be defined as follows.

Definition 4.3.2. $x_i \overset{\alpha_i}{\circ} x_{i+1} \overset{\alpha_{i+1}}{\circ} \cdots \overset{\alpha_{k-2}}{\circ} x_{k-1} \overset{\alpha_{k-1}}{\circ} x_k$, *where* $x_i, x_{i+1}, \ldots, x_{k-1}$, $x_k \in X = (x_1, \ldots, x_n)$ *and* $\alpha_i, \alpha_{i+1}, \ldots, \alpha_{k-2}, \alpha_{k-1} \in \theta = (\alpha_1, \ldots, \alpha_{n-1})$.

$x_i \overset{\alpha_i}{\circ} x_{i+1} \overset{\alpha_{i+1}}{\circ} \cdots \overset{\alpha_{k-2}}{\circ} x_{k-1} \overset{\alpha_{k-1}}{\circ} x_k \overset{def}{=}$ Sub-tree of $Tree_{FP}(X, \theta)$ whose top vertex corresponds to $\overset{\alpha_{max}}{\circ}$, where $\alpha_{max} = max\{\alpha_i,\ \alpha_{i+1}, \ldots, \alpha_{k-2}, \alpha_{k-1}\}$.

Note that the sub-tree defined by Definition 4.3.2 may contain more input nodes other than $x_i, x_{i+1}, \ldots, x_{k-1}, x_k$, for example, in the first figure of Figure 4.3

$$x_2 \overset{3}{\circ} x_3 = x_1 \overset{1}{\circ} x_2 \overset{3}{\circ} x_3 \overset{2}{\circ} x_4 \tag{4.1}$$

4.3.2 Fastest Full-Product Generation Problem(FFP-Problem)

Because most of the vertexes of prefix circuits are $\circ$ operators, it is convenient to quantize time with respect to the delay of this operator. To obtain an input profile, arrival times of n input $x_1 \circ \cdots \circ x_n$ should be normalized with respect to the delay of the $\circ$ operator:

$$t_{x_i} \overset{def}{=} \lceil arrival\ time(x_i)/\Delta \rceil\ (\Delta = delay\ of\ the \circ operator\ and\ i = 1, \ldots, n). \tag{4.2}$$

A normalized input delay profile of the input vector $X = (x_1,\ x_2, \ldots, x_n)$ is defined by t_{x_i}:

$$T_X = (t_{x_1}, t_{x_2}, \ldots, t_{x_n}). \tag{4.3}$$

Let $\tau(\cdot)$ be the delay calculating operator. Since the $\circ$ operator is a binary operator which has unit delay, its operational delay can be expressed as follows,

$$\tau(x_i \circ x_j) = max(t_{x_i}, t_{x_j}) + 1. \tag{4.4}$$

The $\bullet$ operator is introduced to express the time behavior of the $\circ$ operator tree,

$$t_{x_i} \bullet t_{x_j} \stackrel{def}{=} max(t_{x_i}, t_{x_j}) + 1. \tag{4.5}$$

Unlike the $\circ$ operator, the $\bullet$ operator is not associative as shown in Figure 4.4.

Like $\overset{\alpha_i}{\circ}$, to assign an order to the $\bullet$ operator, it is changed into $\overset{\alpha_i}{\bullet}$, where each α_i has distinct integer value from 1 to n-1 that corresponds to the order of the i^{th} operation among the n-1 operations.

f_{FP} which calculates the delay of the FP-Tree of a given input pair (T_X, θ) can be defined as follows.

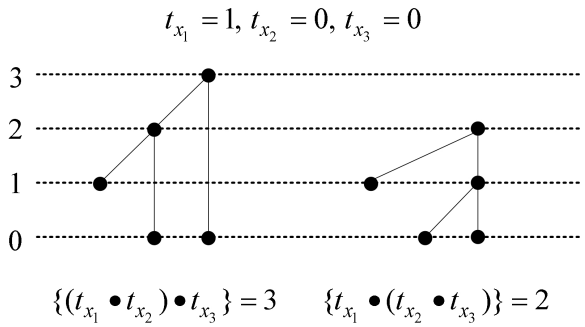

Figure 4.4: Non-associativity example of the $\bullet$ operator.

Definition 4.3.3. *$f_{FP}(T_X, \theta)$, where $X = (x_1, \ldots, x_n)$, $T_X = (t_{x_1}, \ldots, t_{x_n})$, and $\theta = (\alpha_1, \ldots, \alpha_{n-1}) \in \Theta(X)$.*

$$f_{FP}(T_X, \theta) \stackrel{def}{=} t_{x_1} \overset{\alpha_1}{\bullet} t_{x_2} \overset{\alpha_2}{\bullet} \cdots \overset{\alpha_{n-1}}{\bullet} t_{x_n}.$$

By Definition 4.3.1 and 4.3.3,

$$f_{FP}(T_X, \theta) = \tau(Tree_{FP}(X, \theta)) \tag{4.6}$$

Figure 4.5 shows examples of f_{FP} operations.

Definition 4.3.4 is the timing version of Definition 4.3.2.

Definition 4.3.4. $t_{x_i} \overset{\alpha_i}{\bullet} t_{x_{i+1}} \overset{\alpha_{i+1}}{\bullet} \cdots \overset{\alpha_{k-2}}{\bullet} t_{x_{k-1}} \overset{\alpha_{k-1}}{\bullet} tx_k$, *where* $t_{x_i}, t_{x_{i+1}}, \ldots, t_{x_{k-1}}, t_{x_k} \in T_X = (t_{x_1}, \ldots, t_{x_n})$ *and* $\alpha_i, \alpha_{i+1}, \ldots, \alpha_{k-2}, \alpha_{k-1} \in \theta = (\alpha_1, \ldots, \alpha_{n-1})$.

$$t_{x_i} \overset{\alpha_i}{\bullet} t_{x_{i+1}} \overset{\alpha_{i+1}}{\bullet} \cdots \overset{\alpha_{k-2}}{\bullet} t_{x_{k-1}} \overset{\alpha_{k-1}}{\bullet} t_{x_k} \stackrel{def}{=} \tau(x_i \overset{\alpha_i}{\circ} x_{i+1} \overset{\alpha_{i+1}}{\circ} \cdots \overset{\alpha_{k-2}}{\circ} x_{k-1} \overset{\alpha_{k-1}}{\circ} x_k).$$

Finally, the fastest full-product generation problem is defined with f_{FP}.

Definition 4.3.5. *Fastest Full−Product Generation Problem (FFP−Problem).*

For a given input delay profile $T_X = (t_{x_1}, \ldots, t_{x_n})$, find $\theta^* \in \Theta(X)$ such that

$$f_{FP}(T_X, \theta^*) = \min_{\theta \in \Theta(X)} \{f_{FP}(T_X, \theta)\} \tag{4.7}$$

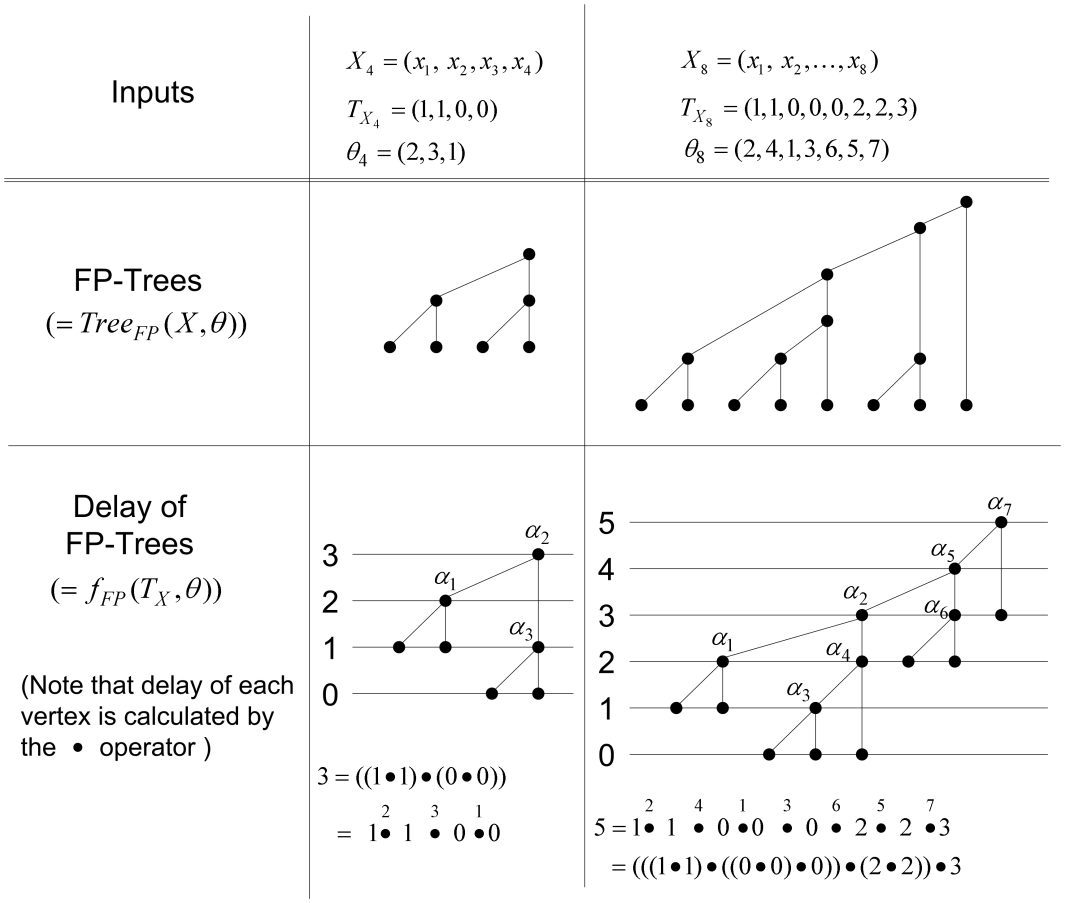

Figure 4.5: 4-bit and 8-bit examples of FP-Tree.

4.3.3 Properties of • Operator

To make an algorithm for the FFP-Problem, the properties of the • operator and FP-Tree need to be examined in advance.

Definition 4.3.6 defines a sub-tree of a given $Tree_{FP}(X, \theta)$ with a given point x_i and operational threshold α_j.

Definition 4.3.6. $Sub_{(X,\theta)}(x_i, \alpha_j)$.

$Sub_{(X,\theta)}(x_i, \alpha_j) \stackrel{def}{=}$ Maximal sub-tree of $Tree_{FP}(X, \theta)$, which includes $x_i \in X$ and is composed of $\alpha_l \in \{\alpha_k | \alpha_k < \alpha_j, \alpha_k \in \theta\}$. If there is no such α_l, $Sub_{(X,\theta)}(x_i, \alpha_j) = x_i$.

Note that by the definition of $\tau(\cdot)$, $\tau(Sub_{(X,\theta)}(x_i, \alpha_j))$ equals to the delay calculation result of $Sub_{(X,\theta)}(x_i, \alpha_j)$.

The following are examples of $Sub_{(X,\theta)}(x_i, \alpha_j)$ from the 8 input FP-Tree of Figure 4.5.

-$Sub_{(X_8,\theta_8)}(x_3, \alpha_5) = x_1 \overset{2}{\circ} x_2 \overset{4}{\circ} x_3 \overset{1}{\circ} x_4 \overset{3}{\circ} x_5$,

$\tau(Sub_{(X_8,\theta_8)}(x_3, \alpha_5)) = (1 \bullet 1) \bullet ((0 \bullet 0) \bullet 0) = 3$

-$Sub_{(X_8,\theta_8)}(x_3, \alpha_2) = x_3 \overset{1}{\circ} x_4 \overset{3}{\circ} x_5$, $\tau(Sub_{(X_8,\theta_8)}(x_3, \alpha_2)) = (0 \bullet 0) \bullet 0 = 2$

-$Sub_{(X_8,\theta_8)}(x_2, \alpha_2) = x_1 \overset{2}{\circ} x_2$, $\tau(Sub_{(X_8,\theta_8)}(x_2, \alpha_2)) = 1 \bullet 1 = 2$

-$Sub_{(X_8,\theta_8)}(x_7, \alpha_2) = x_7$, $\tau(Sub_{(X_8,\theta_8)}(x_7, \alpha_2)) = t_{x_7} = 2$

Now, a tree can be expressed with its sub-trees.

Lemma 4.3.1. $x_i \overset{\alpha_i}{\circ} x_{i+1} = Sub_{(X,\theta)}(x_i, \alpha_i) \overset{\alpha_i}{\circ} Sub_{(X,\theta)}(x_{i+1}, \alpha_i)$ *and* $t_{x_i} \overset{\alpha_i}{\bullet} t_{x_{i+1}} = \tau(Sub_{(X,\theta)}(x_i, \alpha_i)) \bullet \tau(Sub_{(X,\theta)}(x_{i+1}, \alpha_i))$, *where* $t_{x_i}, t_{x_{i+1}} \in T_X$ *and* $\alpha_i \in \theta$ *from a given* $Tree_{FP}(X, \theta)$.

Proof. By Definition 4.3.2, $x_i \overset{\alpha_i}{\circ} x_{i+1}$ is a sub-tree of $Tree_{FP}(X, \theta)$ whose top vertex corresponds to $\overset{\alpha_i}{\circ}$. By Definition 4.3.6, $Sub_{(X,\theta)}(x_i, \alpha_i)$ and $Sub_{(X,\theta)}(x_{i+1}, \alpha_i)$ are the left child and the right child of the top vertex, respectively. Thus,

$$x_i \overset{\alpha_i}{\circ} x_{i+1} = Sub_{(X,\theta)}(x_i, \alpha_i) \overset{\alpha_i}{\circ} Sub_{(X,\theta)}(x_{i+1}, \alpha_i) \tag{4.8}$$

and

$$\tau(x_i \overset{\alpha_i}{\circ} x_{i+1}) = \tau(Sub_{(X,\theta)}(x_i, \alpha_i)) \bullet \tau(Sub_{(X,\theta)}(x_{i+1}, \alpha_i)). \tag{4.9}$$

By Definition 4.3.4,

$$t_{x_i} \overset{\alpha_i}{\bullet} t_{x_{i+1}} = \tau(x_i \overset{\alpha_i}{\circ} x_{i+1}) \tag{4.10}$$

□

The following is an example of Lemma 4.3.1 from the 8-bit FP-Tree of Figure 4.5,

$$\begin{aligned} x_2 \overset{\alpha_2}{\circ} x_3 &= Sub_{(X_8,\theta_8)}(x_2, \alpha_2) \overset{\alpha_2}{\circ} Sub_{(X_8,\theta_8)}(x_3, \alpha_2) = x_1 \overset{2}{\circ} x_2 \overset{\alpha_2}{\circ} x_3 \overset{1}{\circ} x_4 \overset{3}{\circ} x_5 \\ t_{x_2} \overset{\alpha_2}{\bullet} t_{x_3} &= \tau(Sub_{(X_8,\theta_8)}(x_2, \alpha_2)) \bullet \tau(Sub_{(X_8,\theta_8)}(x_3, \alpha_2)) = 2 \bullet 2 = 3. \end{aligned}$$

Definition 4.3.7 is an extension of Definition 4.3.6.

Definition 4.3.7. *$Sub_{(X,\theta)}(T_{sub}, \alpha_j)$ where T_{sub} is a sub − tree of $Tree_{FP}(X, \theta)$.*

$Sub_{(X,\theta)}(T_{sub}, \alpha_j) \stackrel{def}{=}$ Maximal sub-tree of $Tree_{FP}(X, \theta)$, which includes T_{sub} and is composed of $\alpha_l \in \{\alpha_k | \alpha_k < \alpha_j, \alpha_k \in \theta\}$. If there is no such α_l, $Sub_{(X,\theta)}(T_{sub}, \alpha_j) = T_{sub}$.

Lemma 4.3.2 is an extension of Lemma 4.3.1.

Lemma 4.3.2. $x_i \overset{\alpha_i}{\circ} x_{i+1} \overset{\alpha_{i+1}}{\circ} \cdots \overset{\alpha_{i+m-1}}{\circ} x_{i+m} = Sub_{(X,\theta)}(x_i \overset{\alpha_i}{\circ} x_{i+1} \overset{\alpha_{i+1}}{\circ} \cdots \overset{\alpha_{j-1}}{\circ} x_j, \alpha_j) \overset{\alpha_j}{\circ} Sub_{(X,\theta)}(x_{j+1} \overset{\alpha_{j+1}}{\circ} x_{j+2} \overset{\alpha_{j+2}}{\circ} \cdots \overset{\alpha_{i+m-1}}{\circ} x_{i+m}, \alpha_j)$ *and*
$t_{x_i} \overset{\alpha_i}{\circ} t_{x_{i+1}} \overset{\alpha_{i+1}}{\circ} \cdots \overset{\alpha_{i+m-1}}{\circ} t_{x_{i+m}} = \tau(Sub_{(X,\theta)}(x_i \overset{\alpha_i}{\circ} x_{i+1} \overset{\alpha_{i+1}}{\circ} \cdots \overset{\alpha_{j-1}}{\circ} x_j, \alpha_j)) \bullet \tau(Sub_{(X,\theta)}(x_{j+1} \overset{\alpha_{j+1}}{\circ} x_{j+2} \overset{\alpha_{j+2}}{\circ} \cdots \overset{\alpha_{i+m-1}}{\circ} x_{i+m}, \alpha_j))$,
where $\alpha_j = max\{\alpha_i, \ldots, \alpha_{i+m-1}\}$, $t_{x_i}, \ldots, t_{x_{i+m}} \in T_X$ and $\alpha_i, \ldots, \alpha_{i+m-1} \in \theta$ from a given $Tree_{FP}(X, \theta)$.

Proof. This can be proved with similar arguments to the proof of Lemma 4.3.1. □

Lemma 4.3.3, 4.3.4 and 4.3.5 show useful properties of $Sub_{(X,\theta)}(x_i, \alpha_j)$.

Lemma 4.3.3. *For a given $Tree_{FP}(X, \theta)$, if $\alpha_{i-1} < \alpha_i$, then*

$$\tau(Sub_{(X,\theta)}(x_i, \alpha_{i-1})) = t_{x_i}.$$

Proof. x_i is either the right child of the vertex corresponding to $\overset{\alpha_{i-1}}{\circ}$ or the left child of the vertex corresponding to $\overset{\alpha_i}{\circ}$. Because $\alpha_{i-1} < \alpha_i$, by Lemma 4.3.2,

$$x_{i-1} \overset{\alpha_{i-1}}{\circ} x_i \overset{\alpha_i}{\circ} x_{i+1} = Sub_{(X,\theta)}(x_{i-1} \overset{\alpha_{i-1}}{\circ} x_i, \alpha_i) \overset{\alpha_i}{\circ} Sub_{(X,\theta)}(x_{i+1}, \alpha_i). \tag{4.11}$$

Therefore, x_i is the right child of the vertex corresponding to $\overset{\alpha_{i-1}}{\circ}$ and it is connected to the vertex directly. By this fact and Definition 4.3.6, $Sub_{(X,\theta)}(x_i, \alpha_{i-1}) = x_i$ and $\tau(Sub_{(X,\theta)}(x_i, \alpha_{i-1})) = t_{x_i}$.

□

Lemma 4.3.4. *For a given $Tree_{FP}(X, \theta)$, if $\alpha_{i-1} > \alpha_i$, then*

$$\tau(Sub_{(X,\theta)}(x_i, \alpha_i)) = t_{x_i}.$$

Proof. This can be proved with similar arguments to the proof of Lemma 4.3.3 □

Lemma 4.3.5. *For a given $Tree_{FP}(X, \theta)$, if $\alpha_i < \alpha_{i-1}, \alpha_{i+1}$, then*

$$t_{x_i} \overset{\alpha_i}{\bullet} t_{x_{i+1}} = t_{x_i} \bullet t_{x_{i+1}}.$$

Proof. By Lemma 4.3.1, $t_{x_i} \overset{\alpha_i}{\bullet} t_{x_{i+1}} = \tau(Sub_{(X,\theta)}(x_i, \alpha_i)) \bullet \tau(Sub_{(X,\theta)}(x_{i+1}, \alpha_i))$. By Lemma 4.3.4, $\tau(Sub_{(X,\theta)}(x_i, \alpha_i)) = t_{x_i}$ and by Lemma 4.3.3, $\tau(Sub_{(X,\theta)}(x_{i+1}, \alpha_i)) = t_{x_{i+1}}$ □

Lemma 4.3.6 shows the delay shift property of the $\bullet$ operation.

Lemma 4.3.6. *If $t_a \geq t_b$, then $t_a \bullet t_b = t_a \bullet t_a$.*

Proof. $t_a \bullet t_b = \max(t_a, t_b) + 1 = t_a + 1 = \max(t_a, t_a) + 1 = t_a \bullet t_a$. □

Delays of isolated local minimum points can be shifted to the earlier neighbor without increasing the total delay of the FP-Tree. Theorem 4.3.7 presents the formal proof of the delay reserved shift up property of isolated local minimum points of the FP-Tree.

Theorem 4.3.7. *For a given* $T_X = (t_{x_1}, \ldots, t_{x_n})$ *and* $\theta = (\alpha_1, \ldots, \alpha_{n-1}) \in \Theta(X)$,
if $t_{x_i} < t_{x_{i-1}}, t_{x_{i+1}}$ *then* $f_{FP}(T_X, \theta) = f_{FP}(T_X{}', \theta)$,
where $T_X{}' = (t_{x_1}, \ldots, t_{x_{i-1}}, t_{x_i}{}', t_{x_{i+1}}, \ldots, t_{x_n})$ *and* $t_{x_i}{}' = \min(t_{x_{i-1}}, t_{x_{i+1}})$.

Proof. Without loss of generality, assume that

$$t_{x_i} < t_{x_{i-1}} < t_{x_{i+1}} \tag{4.12}$$

Thus,

$$t_{x_i}{}' = min(t_{x_{i-1}}, t_{x_{i+1}}) = t_{x_{i-1}} \tag{4.13}$$

(i)$\alpha_{i-1} < \alpha_i$.

By Lemma 4.3.3

$$t_{x_{i-1}} \overset{\alpha_{i-1}}{\bullet} t_{x_i} = \tau(Sub_{(X,\theta)}(x_{i-1}, \alpha_{i-1})) \bullet t_{x_i} \tag{4.14}$$

and

$$t_{x_{i-1}} \overset{\alpha_{i-1}}{\bullet} t_{x_i}{}' = \tau(Sub_{(X,\theta)}(x_{i-1}, \alpha_{i-1})) \bullet t_{x_i}{}'. \tag{4.15}$$

By Equation (4.12),

$$t_{x_{i-1}} \overset{\alpha_{i-1}}{\bullet} t_{x_i} = \tau(Sub_{(X,\theta)}(x_{i-1}, \alpha_{i-1})) + 1. \tag{4.16}$$

By Equation (4.13),

$$t_{x_{i-1}} \overset{\alpha_{i-1}}{\bullet} t_{x_i}{}' = \tau(Sub_{(X,\theta)}(x_{i-1}, \alpha_{i-1})) + 1. \tag{4.17}$$

Thus,

$$t_{x_{i-1}} \overset{\alpha_{i-1}}{\bullet} t_{x_i}{}' = t_{x_{i-1}} \overset{\alpha_{i-1}}{\bullet} t_{x_i}. \tag{4.18}$$

Because T_X and $T_X{}'$ differ only in t_{x_i} and $t_{x_i}{}'$ and they are applied to the same operation order profile θ, $f_{FP}(T_X, \theta) = f_{FP}(T_X{}', \theta)$.

(ii) $\alpha_i < \alpha_{i-1}$.

Using Lemma 4.3.4 instead of Lemma 4.3.3, this can be proved with similar arguments to (i). □

There are two intuitive optimization schemes for FP-Tree construction. One is to start to make vertexes with the earliest arrival signals as shown in Figure 4.5 and the other is to start from the smallest index as shown in Figure 4.6. These schemes try to reduce the chances of combining two signals whose delay difference is big, which results in wasting time. Based on these two observations, a start bit for FP-Tree generation algorithm is presented.

Definition 4.3.8. *Minimum Delay Smallest Index* ($MD - SI$).

$MD - SI \stackrel{def}{=} \min\{i | t_{x_i} = \min(t_{x_1}, \ldots, t_{x_n})\}$, for a given input delay profile $T_X = (t_{x_1}, \ldots, t_{x_n})$.

After shifting up the isolated minimum delay signals, the nodes near the x_{MD-SI} will have only two patterns which are shown in Figure 4.7.

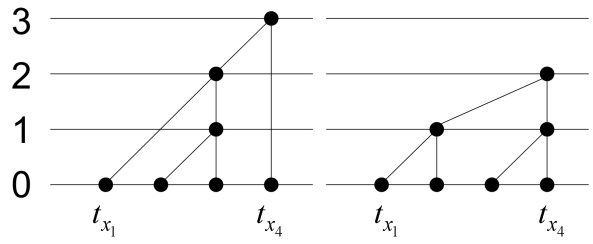

Figure 4.6: Smallest index first optimization scheme example (In the left figure, combining the 2nd and 3rd signals first makes the tree non-optimal).

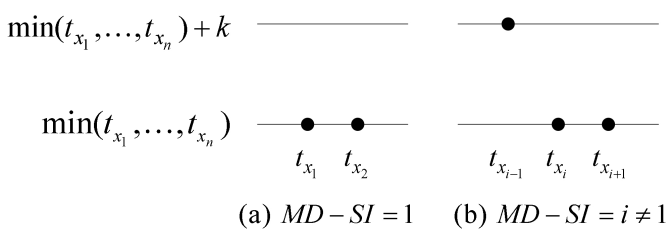

Figure 4.7: Two patterns of the nodes near MD-SI.

Lemma 4.3.8. *Assume that all the isolated local minimum delay signals are removed by the delay reserved shifting of Theorem 4.3.7. Then,*

if MD-SI $= 1$, $t_{x_1} = t_{x_2} = min(t_{x_1}, \ldots, t_{x_n})$, *or*

if MD-SI $= i \neq 1$, $t_{x_i} = t_{x_{i+1}} = min(t_{x_1}, \ldots, t_{x_n}) < t_{x_{i-1}}$.

Proof.

(i) $MD - SI = 1$

Because x_1 cannot be an isolated local minimum signal, $t_{x_1} = t_{x_2}$. Because x_1 is a global minimum delay signal, $t_{x_1} = t_{x_2} = \min(t_{x_1}, \ldots, t_{x_n})$.

(ii) $MD - SI = i \neq 1$

Because x_i cannot be an isolated local minimum signal and it is the smallest index signal among minimum delay signals, $t_{x_{i-1}} \neq t_{x_i} = t_{x_{i+1}}$. Because x_i is a global minimum delay signal, $t_{x_i} = t_{x_{i+1}} = \min(t_{x_1}, \ldots, t_{x_n}) < t_{x_{i-1}}$. □

4.3.4 Fastest Full-Product Generation Algorithm (FFP-Algorithm)

In this section, a greedy algorithm for the FFP-Problem is presented based on the observations in Section 4.3.3. Figure 4.8 shows the algorithm which generates an operation order profile $\theta = (\alpha_1,\ \alpha_2, \ldots, \alpha_{n-1})$ from a given input $X = (x_1,\ x_2, \ldots, x_n)$. Its optimality will be proved later in this section. The run time of FFP-Algorithm is $O(n^2)$ and it requires only $O(n)$ operational memory. An example of FFP-Algorithm is presented in Figure 4.9.

Proposition 4.3.9 and Theorem 4.3.10 proves the optimality of FFP-Algorithm and they require several additional signals and definitions. Let the set of signals of j^{th} iteration of FFP-Algorithm be $X_j = (x_1^j, \ldots, x_{n-j}^j)$ and its delay profile be $T_{X_j} = (t_{x_1^j}, \ldots, t_{x_{n-j}^j})$, then $X_0 = X$ and $x_i^0 = x_i$ and recursive relationships between signals of each iteration as follows:

- $x_i^{j+1} = x_i^j$, $t_{x_i^{j+1}} = t_{x_i^j}$ ($i = 1, \ldots, (MD - SI)_j - 1$)
- $x_i^{j+1} = x_i^j \circ x_{i+1}^j$, $t_{x_i^{j+1}} = t_{x_i^j} \bullet t_{x_{i+1}^j} = t_{x_i^j} + 1$ ($i = (MD - SI)_j$)
- $x_i^{j+1} = x_{i+1}^j$, $t_{x_i^{j+1}} = t_{x_{i+1}^j}$ ($i = (MD - SI)_j + 1, \ldots, n - j - 1$).

Iterative equations above are introduced to exclude the dead signal of each iteration and to make a reduced set of signals for proves. Note that $(MD - SI)_j^X$ in the FFP-Algorithm is about input signal $X = X_0$ throughout all the iterations, but $(MD - SI)_j$ in the recursive equation above is about X_j(set of signals of each iteration) and $(MD - SI)_j^X - (MD - SI)_j$ is the number of dead indexes of X between index 1 and index $(MD - SI)_j^X$.

Algorithm FFP
Input: $X = (x_1,\ x_2, \ldots, x_n)$
Output: $\theta = (\alpha_1,\ \alpha_2, \ldots, \alpha_{n-1})$
begin
1. generate normalized input delay profile $T_X = (t_{x_1}, \ldots, t_{x_n})$ from x_i's
2. set all the indexes live
3. **for** $j = 0 : n-2$
4. perform delay reserved shift up operation on the set of live t_{x_i}'s
   to remove isolated local minimum signals if any
5. find $(MD - SI)_j^X$ among live t_{x_i}'s
6. find $((MD - SI)_j^X + k)$
   (= first live index next to $(MD - SI)_j^X$, $k \geq 1$)
7. make index $(MD - SI)_j^X$ dead and set $t_{x_{(MD-SI)_j^X+k}} = t_{x_{(MD-SI)_j^X}} + 1$
8. $\alpha_{(MD-SI)_j^X} = j + 1$

end

Figure 4.8: Fastest Full-Product Generation Algorithm (FFP-Algorithm).

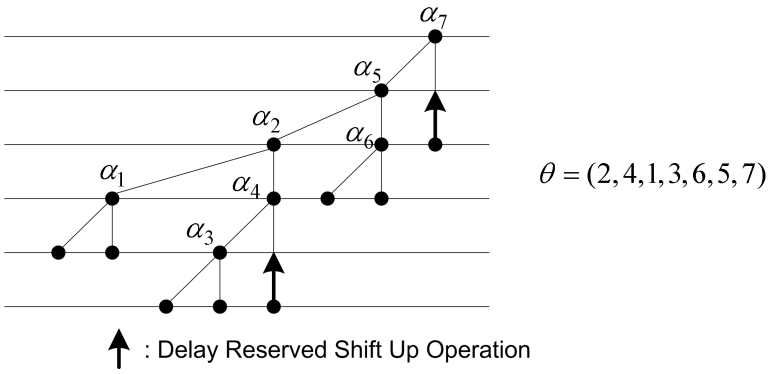

Figure 4.9: FFP-Algorithm example.

Let the optimal delay of FP-Tree of each X_j be τ_j^*, i.e.

$$\tau_j^* = \min_{\theta_j \in \Theta(X_j)} \{f_{FP}(T_{X_j}, \theta_j)\}, \tag{4.19}$$

and the set of all the optimal operation order profiles of X_j be $\Theta^*(X_j)$, i.e.

$$\Theta^*(X_j) = \{\theta | f_{FP}(T_{X_j}, \theta) = \tau_j^*\}, \ where \ j = 0, \ldots, n-2. \tag{4.20}$$

Proposition 4.3.9. *For X_1, there exists a $\theta_1 \in \Theta^*(X_1)$ such that $\tau_0^* = \tau_1^* = f_{FP}(T_{X_1}, \theta_1)$.*

Proof. Assume that there is no such θ_1, i.e.

$$f_{FP}(T_{X_1}, \theta_1) > \tau_0^*, \forall \theta_1 \in \Theta(X_1). \tag{4.21}$$

By Lemma 4.3.8, there exist only two cases, $(MD - SI)_0 = 1$ and $(MD - SI)_0 = k \neq 0$.

Let an optimal FP-Tree of X_0 be $Tree_{FP}(X_0, \theta_0^*)$, where $\theta_0^* = (\alpha_1, \ldots, \alpha_{n-1}) \in \Theta^*(X_0)$.

Case 1) $(MD - SI)_0 = 1$

$$t_{x_1^0} = t_{x_2^0} \leq t_{x_3^0}. \tag{4.22}$$

Equation (4.21) means that none of the FP-Trees of X_1 can be optimal, which implies that none of the optimal FP-Trees of X_0 should have the vertex that is made up of x_1^0 and x_2^0. Therefore, following Statement 1 can be addressed.

Statement 1: *$\alpha_1 > \alpha_2$, in the sub-tree $x_1^0 \overset{\alpha_1}{\circ} x_2^0 \overset{\alpha_2}{\circ} x_3^0$ of all the optimal FP-Trees of X_0.*

By Lemma 4.3.2,

$$\left[t_{x_1^0} \overset{\alpha_1}{\bullet} t_{x_2^0} \overset{\alpha_2}{\bullet} t_{x_3^0}\right]^{\theta_0^*}_{\alpha_1>\alpha_2} = t_{x_1^0} \bullet \tau\left(Sub_{(X_0,\theta_0^*)}(x_2^0 \overset{\alpha_2}{\circ} x_3^0, \alpha_1)\right). \tag{4.23}$$

Exchange the location of α_1 and α_2 of θ_0^* and let the changed operation order profile be $\theta_0{}'$, then

$$\left[t_{x_1^0} \overset{\alpha_2}{\bullet} t_{x_2^0} \overset{\alpha_1}{\bullet} t_{x_3^0}\right]^{\theta_0{}'}_{\alpha_1>\alpha_2} = (t_{x_1^0} \bullet t_{x_2^0}) \bullet \tau\left(Sub_{(X_0,\theta_0{}')}(x_3^0, \alpha_1)\right). \tag{4.24}$$

Because $Sub_{(X_0,\theta_0^*)}(x_2^0 \overset{\alpha_2}{\circ} x_3^0, \alpha_1)$ can be constructed by inserting a vertex into $Sub_{(X_0,\theta_0{}')}(x_3^0, \alpha_1)$ with x_2^0 and $\overset{\alpha_2}{\circ}$,

$$\tau\left(Sub_{(X_0,\theta_0{}')}(x_3^0, \alpha_1)\right) \leq \tau\left(Sub_{(X_0,\theta_0^*)}(x_2^0 \overset{\alpha_2}{\circ} x_3^0, \alpha_1)\right). \tag{4.25}$$

By Equation (4.22),

$$\left[t_{x_1^0} \overset{\alpha_2}{\bullet} t_{x_2^0}\right]^{\theta_0{}'} \leq \tau\left(Sub_{(X_0,\theta_0^*)}(x_2^0 \overset{\alpha_2}{\circ} x_3^0, \alpha_1)\right). \tag{4.26}$$

By Equations (4.25) and (4.26),

$$\left[t_{x_1^0} \overset{\alpha_2}{\bullet} t_{x_2^0} \overset{\alpha_1}{\bullet} t_{x_3^0}\right]^{\theta_0{}'}_{\alpha_1>\alpha_2} \leq \left[t_{x_1^0} \overset{\alpha_1}{\bullet} t_{x_2^0} \overset{\alpha_2}{\bullet} t_{x_3^0}\right]^{\theta_0^*}_{\alpha_1>\alpha_2} \tag{4.27}$$

and this contradicts statement 1.

Case 2) $(MD - SI)_0 = k \neq 1$

$$t_{x_{k-1}^0} > t_{x_k^0} = t_{x_{k+1}^0} \leq t_{x_{k+2}^0}. \tag{4.28}$$

Equation (4.21) implies that none of the optimal FP-Trees of X_0 should have the vertex that is made up of x_k^0 and x_{k+1}^0. Therefore, following Statement 2 can be addressed.

Statement 2: *In the sub-tree* $x^0_{k-1} \overset{\alpha_{k-1}}{\circ} x^0_k \overset{\alpha_k}{\circ} x^0_{k+1} \overset{\alpha_{k+1}}{\circ} x^0_{k+2}$ *of all the optimal FP-Trees of* X_0, $\sim(\alpha_k < \alpha_{k-1}, \alpha_{k+1})$, *where '*$\sim$*' is logical negation, i.e.*

(i) $\alpha_{k-1} < \alpha_k < \alpha_{k+1}$ *or*

(ii) $\alpha_{k+1} < \alpha_k < \alpha_{k-1}$ *or*

(iii) $\alpha_{k+1} < \alpha_{k-1} < \alpha_k$ *or*

(iv) $\alpha_{k-1} < \alpha_{k+1} < \alpha_k$

(i) $\alpha_{k-1} < \alpha_k < \alpha_{k+1}$

By Lemma 4.3.2,

$$\left[t_{x^0_{k-1}} \overset{\alpha_{k-1}}{\bullet} t_{x^0_k} \overset{\alpha_k}{\bullet} t_{x^0_{k+1}} \overset{\alpha_{k+1}}{\bullet} t_{x^0_{k+2}}\right]^{\theta_0^*}_{\alpha_{k-1}<\alpha_k<\alpha_{k+1}}$$
$$= \tau\left(Sub_{(X_0,\theta_0^*)}(x^0_{k-1} \overset{\alpha_{k-1}}{\circ} x^0_k \overset{\alpha_k}{\circ} x^0_{k+1}, \alpha_{k+1})\right) \bullet \tau\left(Sub_{(X_0,\theta_0^*)}(x^0_{k+2}, \alpha_{k+1})\right). \quad (4.29)$$

Exchange the location of α_k and α_{k-1} of θ_0^* and let the changed operation order profile be $\theta_0{}'$, then

$$\left[t_{x^0_{k-1}} \overset{\alpha_k}{\bullet} t_{x^0_k} \overset{\alpha_{k-1}}{\bullet} t_{x^0_{k+1}} \overset{\alpha_{k+1}}{\bullet} t_{x^0_{k+2}}\right]^{\theta_0{}'}_{\alpha_{k-1}<\alpha_k<\alpha_{k+1}}$$
$$= \tau\left(Sub_{(X_0,\theta_0{}')}(x^0_{k-1} \overset{\alpha_k}{\circ} x^0_k \overset{\alpha_{k-1}}{\circ} x^0_{k+1}, \alpha_{k+1})\right) \bullet \tau\left(Sub_{(X_0,\theta_0{}')}(x^0_{k+2}, \alpha_{k+1})\right). \quad (4.30)$$

By Lemma 4.3.2 and Lemma 4.3.3,

$$\left[t_{x^0_{k-1}} \overset{\alpha_{k-1}}{\bullet} t_{x^0_k} \overset{\alpha_k}{\bullet} t_{x^0_{k+1}}\right]^{\theta_0^*}_{\alpha_{k-1}<\alpha_k<\alpha_{k+1}}$$
$$= \tau\left(Sub_{(X_0,\theta_0^*)}(x^0_{k-1} \overset{\alpha_{k-1}}{\circ} x^0_k, \alpha_k)\right) \bullet t_{x^0_{k+1}}. \quad (4.31)$$

By Lemma 4.3.2 and Lemma 4.3.5,

$$\left[t_{x_{k-1}^0} \overset{\alpha_k}{\bullet} t_{x_k^0} \overset{\alpha_{k-1}}{\bullet} t_{x_{k+1}^0}\right]^{\theta_0'}_{\alpha_{k-1}<\alpha_k<\alpha_{k+1}} = \tau\left(Sub_{(X_0,\theta_0')}(x_{k-1}^0, \alpha_k)\right) \bullet (t_{x_k^0} \bullet t_{x_{k+1}^0}). \quad (4.32)$$

By Equation (4.28) and because $Sub_{(X_0,\theta_0^*)}(x_{k-1}^0 \overset{\alpha_{k-1}}{\circ} x_k^0, \alpha_k)$ can be constructed by inserting a vertex into $Sub_{(X_0,\theta_0')}(x_{k-1}^0, \alpha_k)$ with x_k^0 and $\overset{\alpha_{k-1}}{\circ}$,

$$t_{x_k^0} \bullet t_{x_{k+1}^0} \leq t_{x_{k-1}^0} \leq \tau\left(Sub_{(X_0,\theta_0')}(x_{k-1}^0, \alpha_k)\right) \leq \tau\left(Sub_{(X_0,\theta_0^*)}(x_{k-1}^0 \overset{\alpha_{k-1}}{\circ} x_k^0, \alpha_k)\right) \quad (4.33)$$

and therefore,

$$\left[t_{x_{k-1}^0} \overset{\alpha_k}{\bullet} t_{x_k^0} \overset{\alpha_{k-1}}{\bullet} t_{x_{k+1}^0}\right]^{\theta_0'}_{\alpha_{k-1}<\alpha_k<\alpha_{k+1}} \leq \left[t_{x_{k-1}^0} \overset{\alpha_{k-1}}{\bullet} t_{x_k^0} \overset{\alpha_k}{\bullet} t_{x_{k+1}^0}\right]^{\theta_0^*}_{\alpha_{k-1}<\alpha_k<\alpha_{k+1}}. \quad (4.34)$$

Thus,

$$\left[t_{x_{k-1}^0} \overset{\alpha_k}{\bullet} t_{x_k^0} \overset{\alpha_{k-1}}{\bullet} t_{x_{k+1}^0} \overset{\alpha_{k+1}}{\bullet} t_{x_{k+2}^0}\right]^{\theta_0'}_{\alpha_{k-1}<\alpha_k<\alpha_{k+1}} \leq \left[t_{x_{k-1}^0} \overset{\alpha_{k-1}}{\bullet} t_{x_k^0} \overset{\alpha_k}{\bullet} t_{x_{k+1}^0} \overset{\alpha_{k+1}}{\bullet} t_{x_{k+2}^0}\right]^{\theta_0^*}_{\alpha_{k-1}<\alpha_k<\alpha_{k+1}}. \quad (4.35)$$

(ii) $\alpha_{k+1} < \alpha_k < \alpha_{k-1}$

By Lemma 4.3.2,

$$\left[t_{x_{k-1}^0} \overset{\alpha_{k-1}}{\bullet} t_{x_k^0} \overset{\alpha_k}{\bullet} t_{x_{k+1}^0} \overset{\alpha_{k+1}}{\bullet} t_{x_{k+2}^0}\right]^{\theta_0^*}_{\alpha_{k+1}<\alpha_k<\alpha_{k-1}} = \tau\left(Sub_{(X_0,\theta_0^*)}(x_{k-1}^0, \alpha_{k-1})\right) \bullet \tau\left(Sub_{(X_0,\theta_0^*)}(x_k^0 \overset{\alpha_k}{\circ} x_{k+1}^0 \overset{\alpha_{k+1}}{\circ} x_{k+2}^0, \alpha_{k-1})\right). \quad (4.36)$$

Exchange the location of α_k and α_{k+1} of θ_0^* and let the changed operation order profile be $\theta_0{}'$, then

$$\left[t_{x_{k-1}^0} \overset{\alpha_{k-1}}{\bullet} t_{x_k^0} \overset{\alpha_{k+1}}{\bullet} t_{x_{k+1}^0} \overset{\alpha_k}{\bullet} t_{x_{k+2}^0}\right]^{\theta_0{}'}_{\alpha_{k+1}<\alpha_k<\alpha_{k-1}} = \tau\left(Sub_{(X_0,\theta_0{}')}(x_{k-1}^0, \alpha_{k-1})\right) \bullet \tau\left(Sub_{(X_0,\theta_0{}')}(x_k^0 \overset{\alpha_{k+1}}{\circ} x_{k+1}^0 \overset{\alpha_k}{\circ} x_{k+2}^0, \alpha_{k-1})\right). \tag{4.37}$$

By Lemma 4.3.2 and Lemma 4.3.4,

$$\left[t_{x_k^0} \overset{\alpha_k}{\bullet} t_{x_{k+1}^0} \overset{\alpha_{k+1}}{\bullet} t_{x_{k+2}^0}\right]^{\theta_0^*}_{\alpha_{k+1}<\alpha_k<\alpha_{k-1}} = t_{x_k^0} \bullet \tau\left(Sub_{(X_0,\theta_0^*)}(x_{k+1}^0 \overset{\alpha_{k+1}}{\circ} x_{k+2}^0, \alpha_k)\right). \tag{4.38}$$

By Lemma 4.3.2 and Lemma 4.3.5,

$$\left[t_{x_k^0} \overset{\alpha_{k+1}}{\bullet} t_{x_{k+1}^0} \overset{\alpha_k}{\bullet} t_{x_{k+2}^0}\right]^{\theta_0{}'}_{\alpha_{k+1}<\alpha_k<\alpha_{k-1}} = (t_{x_k^0} \bullet t_{x_{k+1}^0}) \bullet \tau\left(Sub_{(X_0,\theta_0{}')}(x_{k+2}^0, \alpha_k)\right). \tag{4.39}$$

By Equation (4.28) and because $Sub_{(X_0,\theta_0^*)}(x_{k+1}^0 \overset{\alpha_{k+1}}{\circ} x_{k+2}^0, \alpha_k)$ can be constructed by inserting a vertex into $Sub_{(X_0,\theta_0{}')}(x_{k+2}^0, \alpha_k)$ with x_{k+1}^0 and $\overset{\alpha_{k+1}}{\circ}$,

$$(t_{x_k^0} \bullet t_{x_{k+1}^0}), \tau\left(Sub_{(X_0,\theta_0{}')}(x_{k+2}^0, \alpha_k)\right) \leq \tau\left(Sub_{(X_0,\theta_0^*)}(x_{k+1}^0 \overset{\alpha_{k+1}}{\circ} x_{k+2}^0, \alpha_k)\right) \tag{4.40}$$

and therefore,

$$\left[t_{x_k^0} \overset{\alpha_{k+1}}{\bullet} t_{x_{k+1}^0} \overset{\alpha_k}{\bullet} t_{x_{k+2}^0}\right]^{\theta_0{}'}_{\alpha_{k+1}<\alpha_k<\alpha_{k-1}} \leq \left[t_{x_k^0} \overset{\alpha_k}{\bullet} t_{x_{k+1}^0} \overset{\alpha_{k+1}}{\bullet} t_{x_{k+2}^0}\right]^{\theta_0^*}_{\alpha_{k+1}<\alpha_k<\alpha_{k-1}}. \tag{4.41}$$

Thus,

$$\left[t_{x_{k-1}^0} \overset{\alpha_{k-1}}{\bullet} t_{x_k^0} \overset{\alpha_{k+1}}{\bullet} t_{x_{k+1}^0} \overset{\alpha_k}{\bullet} t_{x_{k+2}^0}\right]^{\theta_0{}'}_{\alpha_{k+1}<\alpha_k<\alpha_{k-1}} \leq \left[t_{x_{k-1}^0} \overset{\alpha_{k-1}}{\bullet} t_{x_k^0} \overset{\alpha_k}{\bullet} t_{x_{k+1}^0} \overset{\alpha_{k+1}}{\bullet} t_{x_{k+2}^0}\right]^{\theta_0^*}_{\alpha_{k+1}<\alpha_k<\alpha_{k-1}}. \tag{4.42}$$

(iii) $\alpha_{k+1} < \alpha_{k-1} < \alpha_k$

By Lemma 4.3.2,

$$\left[t_{x_{k-1}^0} \overset{\alpha_{k-1}}{\bullet} t_{x_k^0} \overset{\alpha_k}{\bullet} t_{x_{k+1}^0} \overset{\alpha_{k+1}}{\bullet} t_{x_{k+2}^0}\right]^{\theta_0^*}_{\alpha_{k+1}<\alpha_{k-1}<\alpha_k}$$
$$= \tau\left(Sub_{(X_0,\theta_0^*)}(x_{k-1}^0 \overset{\alpha_{k-1}}{\circ} x_k^0, \alpha_k)\right) \bullet \tau\left(Sub_{(X_0,\theta_0^*)}(x_{k+1}^0 \overset{\alpha_{k+1}}{\circ} x_{k+2}^0, \alpha_k)\right). \quad (4.43)$$

Exchange the location of α_k and α_{k+1} of θ_0^* and let the changed operation order profile be $\theta_0{}'$, then

$$\left[t_{x_{k-1}^0} \overset{\alpha_{k-1}}{\bullet} t_{x_k^0} \overset{\alpha_{k+1}}{\bullet} t_{x_{k+1}^0} \overset{\alpha_k}{\bullet} t_{x_{k+2}^0}\right]^{\theta_0{}'}_{\alpha_{k+1}<\alpha_{k-1}<\alpha_k}$$
$$= \tau\left(Sub_{(X_0,\theta_0{}')}(x_{k-1}^0 \overset{\alpha_{k-1}}{\circ} x_k^0 \overset{\alpha_{k+1}}{\circ} x_{k+1}^0, \alpha_k)\right) \bullet \tau\left(Sub_{(X_0,\theta_0{}')}(x_{k+2}^0, \alpha_k)\right). \quad (4.44)$$

By Lemma 4.3.1, Lemma 4.3.3 and Equation (4.28),

$$\left[t_{x_{k-1}^0} \overset{\alpha_{k-1}}{\bullet} t_{x_k^0}\right]^{\theta_0^*}_{\alpha_{k+1}<\alpha_{k-1}<\alpha_k}$$
$$= \tau\left(Sub_{(X_0,\theta_0^*)}(x_{k-1}^0, \alpha_{k-1})\right) \bullet t_{x_k^0} = \tau\left(Sub_{(X_0,\theta_0^*)}(x_{k-1}^0, \alpha_{k-1})\right) + 1. \quad (4.45)$$

By Lemma 4.3.2, Lemma 4.3.5 and Equation (4.28),

$$\left[t_{x_{k-1}^0} \overset{\alpha_{k-1}}{\bullet} t_{x_k^0} \overset{\alpha_{k+1}}{\bullet} t_{x_{k+1}^0}\right]^{\theta_0{}'}_{\alpha_{k+1}<\alpha_{k-1}<\alpha_k}$$
$$= \tau\left(Sub_{(X_0,\theta_0{}')}(x_{k-1}^0, \alpha_{k-1})\right) \bullet (t_{x_k^0} \bullet t_{x_{k+1}^0}) = \tau\left(Sub_{(X_0,\theta_0{}')}(x_{k-1}^0, \alpha_{k-1})\right) + 1. \quad (4.46)$$

Because $Sub_{(X_0,\theta_0^*)}(x_{k-1}^0, \alpha_{k-1})$ is not changed by exchanging the location of α_k and α_{k+1}, $Sub_{(X_0,\theta_0{}')}(x_{k-1}^0, \alpha_{k-1})$ and $Sub_{(X_0,\theta_0^*)}(x_{k-1}^0, \alpha_{k-1})$ are the same. Thus,

$$\left[t_{x_{k-1}^0} \overset{\alpha_{k-1}}{\bullet} t_{x_k^0} \overset{\alpha_{k+1}}{\bullet} t_{x_{k+1}^0}\right]^{\theta_0{}'}_{\alpha_{k+1}<\alpha_{k-1}<\alpha_k} = \left[t_{x_{k-1}^0} \overset{\alpha_{k-1}}{\bullet} t_{x_k^0}\right]^{\theta_0^*}_{\alpha_{k+1}<\alpha_{k-1}<\alpha_k} \quad (4.47)$$

and therefore, because $Sub_{(X_0,\theta_0{}')}(x^0_{k-1} \overset{\alpha_{k-1}}{\circ} x^0_k \overset{\alpha_{k+1}}{\circ} x^0_{k+1}, \alpha_k)$ and $Sub_{(X_0,\theta^*_0)}(x^0_{k-1} \overset{\alpha_{k-1}}{\circ} x^0_k, \alpha_k)$ are identical except their sub-trees, $x^0_{k-1} \overset{\alpha_{k-1}}{\circ} x^0_k \overset{\alpha_{k+1}}{\circ} x^0_{k+1}$ and $x^0_{k-1} \overset{\alpha_{k-1}}{\circ} x^0_k$,

$$\tau\left(Sub_{(X_0,\theta_0{}')}(x^0_{k-1} \overset{\alpha_{k-1}}{\circ} x^0_k \overset{\alpha_{k+1}}{\circ} x^0_{k+1}, \alpha_k)\right) = \tau\left(Sub_{(X_0,\theta^*_0)}(x^0_{k-1} \overset{\alpha_{k-1}}{\circ} x^0_k, \alpha_k)\right). \quad (4.48)$$

Because $Sub_{(X_0,\theta^*_0)}(x^0_{k+1} \overset{\alpha_{k+1}}{\circ} x^0_{k+2}, \alpha_k)$ can be constructed by inserting a vertex into $Sub_{(X_0,\theta_0{}')}(x^0_{k+2}, \alpha_k)$ with x^0_{k+1} and $\overset{\alpha_{k+1}}{\circ}$,

$$\tau\left(Sub_{(X_0,\theta_0{}')}(x^0_{k+2}, \alpha_k)\right) \leq \tau\left(Sub_{(X_0,\theta^*_0)}(x^0_{k+1} \overset{\alpha_{k+1}}{\circ} x^0_{k+2}, \alpha_k)\right). \quad (4.49)$$

By Equations (4.48) and (4.49),

$$\left[t_{x^0_{k-1}} \overset{\alpha_{k-1}}{\bullet} t_{x^0_k} \overset{\alpha_{k+1}}{\bullet} t_{x^0_{k+1}} \overset{\alpha_k}{\bullet} t_{x^0_{k+2}}\right]^{\theta_0{}'}_{\alpha_{k+1}<\alpha_{k-1}<\alpha_k} \leq \left[t_{x^0_{k-1}} \overset{\alpha_{k-1}}{\bullet} t_{x^0_k} \overset{\alpha_k}{\bullet} t_{x^0_{k+1}} \overset{\alpha_{k+1}}{\bullet} t_{x^0_{k+2}}\right]^{\theta^*_0}_{\alpha_{k+1}<\alpha_{k-1}<\alpha_k}. \quad (4.50)$$

(iv) $\alpha_{k-1} < \alpha_{k+1} < \alpha_k$

By Lemma 4.3.2,

$$\left[t_{x^0_{k-1}} \overset{\alpha_{k-1}}{\bullet} t_{x^0_k} \overset{\alpha_k}{\bullet} t_{x^0_{k+1}} \overset{\alpha_{k+1}}{\bullet} t_{x^0_{k+2}}\right]^{\theta^*_0}_{\alpha_{k-1}<\alpha_{k+1}<\alpha_k} = \tau\left(Sub_{(X_0,\theta^*_0)}(x^0_{k-1} \overset{\alpha_{k-1}}{\circ} x^0_k, \alpha_k)\right) \bullet \tau\left(Sub_{(X_0,\theta^*_0)}(x^0_{k+1} \overset{\alpha_{k+1}}{\circ} x^0_{k+2}, \alpha_k)\right). \quad (4.51)$$

Let $\alpha_{k-1} < \beta_1 < \beta_2 < \cdots < \beta_p < \alpha_{k+1} < \gamma_1 < \cdots < \gamma_q < \alpha_k \in \theta^*_0$ denote all the elements of operation order profile which correspond to the vertexes of sub-tree $x^0_{k-1} \overset{\alpha_{k-1}}{\circ} x^0_k \overset{\alpha_k}{\circ} x^0_{k+1} \overset{\alpha_{k+1}}{\circ} x^0_{k+2}$ between α_{k-1} and α_k. Let prime denote the changed

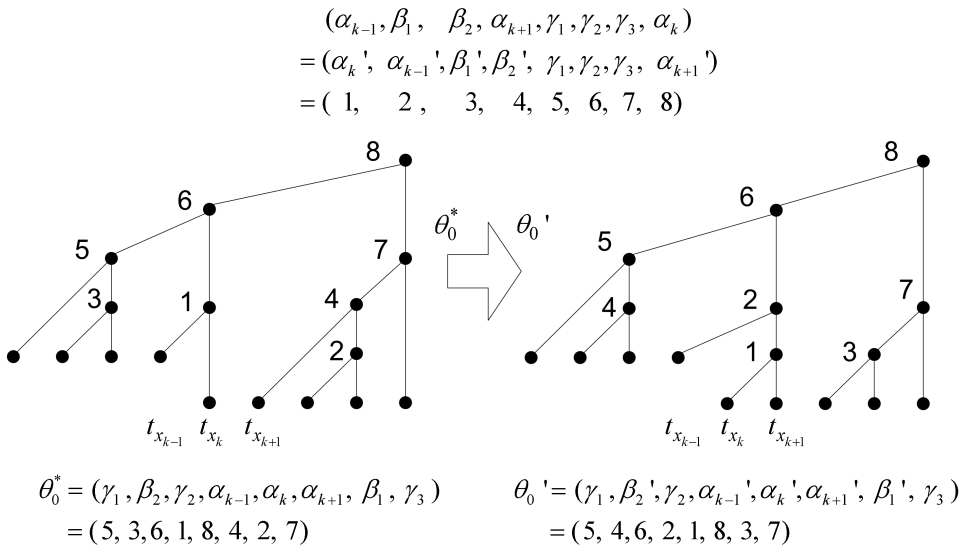

Figure 4.10: An example of Equation (4.52).

elements of θ_0^* by the following equation and $\theta_0{}'$ denotes the changed operation order profile (an example is shown in Figure 4.10),

$$\begin{array}{ccccccccccccc} \alpha_k{}' & < & \alpha_{k-1}{}' & < & \beta_1{}' & < \cdots < & \beta_{p-1}{}' & < & \beta_p{}' & < \gamma_1 < \cdots < \gamma_q < & \alpha_{k+1}{}' \\ \| & & \| & & \| & \cdots & \| & & \| & & \| \\ \alpha_{k-1} & < & \beta_1 & < & \beta_2 & < \cdots < & \beta_p & < & \alpha_{k+1} & < \gamma_1 < \cdots < \gamma_q < & \alpha_k. \end{array} \tag{4.52}$$

then,

$$\begin{aligned} &\left[t_{x_{k-1}^0} \overset{\alpha_{k-1}{}'}{\bullet} t_{x_k^0} \overset{\alpha_k{}'}{\bullet} t_{x_{k+1}^0} \overset{\alpha_{k+1}{}'}{\bullet} t_{x_{k+2}^0} \right]^{\theta_0{}'}_{\alpha_k{}' < \alpha_{k-1}{}' < \alpha_{k+1}{}'} \\ &= \tau\left(Sub_{(X_0, \theta_0{}')}(x_{k-1}^0 \overset{\alpha_{k-1}{}'}{\circ} x_k^0 \overset{\alpha_k{}'}{\circ} x_{k+1}^0, \alpha_{k+1}{}') \right) \bullet \tau\left(Sub_{(X_0, \theta_0{}')}(x_{k+2}^0, \alpha_{k+1}{}') \right). \end{aligned} \tag{4.53}$$

By Lemma 4.3.1, Lemma 4.3.3 and Equation (4.28),

$$\left[t_{x_{k-1}^0} \overset{\alpha_{k-1}}{\bullet} t_{x_k^0}\right]^{\theta_0^*}_{\alpha_{k-1}<\alpha_{k+1}<\alpha_k} = \tau\left(Sub_{(X_0,\theta_0^*)}(x_{k-1}^0, \alpha_{k-1})\right) \bullet t_{x_k^0} = \tau\left(Sub_{(X_0,\theta_0^*)}(x_{k-1}^0, \alpha_{k-1})\right) + 1. \quad (4.54)$$

By Lemma 4.3.2, Lemma 4.3.5 and Equation (4.28),

$$\left[t_{x_{k-1}^0} \overset{\alpha_{k-1}{}'}{\bullet} t_{x_k^0} \overset{\alpha_k{}'}{\bullet} t_{x_{k+1}^0}\right]^{\theta_0{}'}_{\alpha_k{}'<\alpha_{k-1}{}'<\alpha_{k+1}{}'} = \tau\left(Sub_{(X_0,\theta_0{}')}(x_{k-1}^0, \alpha_{k-1}{}')\right) \bullet (t_{x_k^0} \bullet t_{x_{k+1}^0}) = \tau\left(Sub_{(X_0,\theta_0{}')}(x_{k-1}^0, \alpha_{k-1}{}')\right) + 1. \quad (4.55)$$

Because $Sub_{(X_0,\theta_0{}')}(x_{k-1}^0, \alpha_{k-1}{}')$ doesn't include the vertex corresponding to $\overset{\alpha_k{}'}{\circ}$, $Sub_{(X_0,\theta_0{}')}(x_{k-1}^0, \alpha_{k-1}{}')$ and $Sub_{(X_0,\theta_0^*)}(x_{k-1}^0, \alpha_{k-1})$ are the same by Equation (4.52). Thus,

$$\left[t_{x_{k-1}^0} \overset{\alpha_{k-1}{}'}{\bullet} t_{x_k^0} \overset{\alpha_k{}'}{\bullet} t_{x_{k+1}^0}\right]^{\theta_0{}'}_{\alpha_k{}'<\alpha_{k-1}{}'<\alpha_{k+1}{}'} = \left[t_{x_{k-1}^0} \overset{\alpha_{k-1}}{\bullet} t_{x_k^0}\right]^{\theta_0^*}_{\alpha_{k-1}<\alpha_{k+1}<\alpha_k} \quad (4.56)$$

and therefore, because $Sub_{(X_0,\theta_0{}')}(x_{k-1}^0 \overset{\alpha_{k-1}{}'}{\circ} x_k^0 \overset{\alpha_k{}'}{\circ} x_{k+1}^0, \alpha_{k+1}{}')$ and $Sub_{(X_0,\theta_0^*)}(x_{k-1}^0 \overset{\alpha_{k-1}}{\circ} x_k^0, \alpha_k)$ are identical except their sub-trees, $x_{k-1}^0 \overset{\alpha_{k-1}{}'}{\circ} x_k^0 \overset{\alpha_k{}'}{\circ} x_{k+1}^0$ and $x_{k-1}^0 \overset{\alpha_{k-1}}{\circ} x_k^0$,

$$\tau\left(Sub_{(X_0,\theta_0{}')}(x_{k-1}^0 \overset{\alpha_{k-1}{}'}{\circ} x_k^0 \overset{\alpha_k{}'}{\circ} x_{k+1}^0, \alpha_{k+1}{}')\right) = \tau\left(Sub_{(X_0,\theta_0^*)}(x_{k-1}^0 \overset{\alpha_{k-1}}{\circ} x_k^0, \alpha_k)\right). \quad (4.57)$$

If the vertex corresponding to $\overset{\alpha_{k+1}}{\circ}$ is removed from $Sub_{(X_0,\theta_0^*)}(x_{k+1}^0 \overset{\alpha_{k+1}}{\circ} x_{k+2}^0, \alpha_k)$, the sub-tree becomes identical with $Sub_{(X_0,\theta_0{}')}(x_{k+2}^0, \alpha_{k+1}{}')$ by Equation (4.52) and therefore,

$$\tau\left(Sub_{(X_0,\theta_0{}')}(x_{k+2}^0, \alpha_{k+1}{}')\right) \leq \tau\left(Sub_{(X_0,\theta_0^*)}(x_{k+1}^0 \overset{\alpha_{k+1}}{\circ} x_{k+2}^0, \alpha_k)\right). \quad (4.58)$$

By Equations (4.57) and (4.58),

$$t_{x_{k-1}^0} \overset{\alpha_{k-1}'}{\bullet} t_{x_k^0} \overset{\alpha_k'}{\bullet} t_{x_{k+1}^0} \overset{\alpha_{k+1}'}{\bullet} t_{x_{k+2}^0} \leq t_{x_{k-1}^0} \overset{\alpha_{k-1}}{\bullet} t_{x_k^0} \overset{\alpha_k}{\bullet} t_{x_{k+1}^0} \overset{\alpha_{k+1}}{\bullet} t_{x_{k+2}^0}. \tag{4.59}$$

(i), (ii), (iii) and (iv) contradict Statement 2. □

Theorem 4.3.10. *The operation order profile θ_{FFP} from FFP-Algorithm is an optimal solution for the FFP-Problem on input X, i.e.*

$$f_{FP}(T_X, \theta_{FFP}) = \min_{\theta \in \Theta(X)} \{f_{FP}(T_X, \theta)\}$$

Proof. Proposition 4.3.9 proves

$$\tau_0^* = \tau_1^* = \min_{\theta \in \Theta(X_1)} \{f_{FP}(T_{X_1}, \theta)\}. \tag{4.60}$$

If proposition 4.3.9 is applied again on X_2, it proves

$$\tau_1^* = \tau_2^* = \min_{\theta \in \Theta(X_2)} \{f_{FP}(T_{X_2}, \theta)\}. \tag{4.61}$$

Thus, by applying Proposition 4.3.9 repeatedly, $\tau_0^* = \tau_1^* = \cdots = {\tau_{n-2}}^*$ can be proved. □

4.4 Prefix Circuit Design Using the FFP-Algorithm

In Section 4.3, only the design method of the FP-Tree was examined. However, to make a complete prefix circuit, SP-Trees (see Figure 4.2) should also be constructed. Since basic prefix schemes and various hybrid schemes can be used for prefix circuit design according to the patterns of input profile as proposed in Chapter 3, they are also applicable to the design of SP-Trees. Among them, only the minimum depth Ladner-Fischer scheme [2] will be used for SP-Trees in this dissertation for simplicity. One of the advantages of the Ladner-Fischer scheme is that it is easy to combine with the FFP-Algorithm to make an algorithm for designing prefix circuit. Let FFP-Ladner-Fischer Algorithm denote the prefix circuit design algorithm which uses the FFP-Algorithm for the FP-Tree and the Ladner-Fischer for the SP-Trees as shown in Figure 4.11. The algorithm generates a prefix circuit from a given input $X = (x_1, x_2, \ldots, x_n)$. In the algorithm, the inner for-loop implements the Ladner-Fischer scheme for the SP-Trees. An example of FFP-Ladner-Fischer Algorithm is presented in Figure 4.12.

Size and delay will be considered as performance measures on a product circuit. The size is the number of product vertexes in the circuit. For the delay, the delay of the FP-Tree and the lateral fan-out of the entire circuit are considered. The delay of the FP-Tree of FFP-Ladner-Fischer tree is $f_{FP}(T_X, \theta_{FFP})$ from Theorem 4.3.10 and this delay is not increased by the SP-Trees if the effect of fan-out is neglected (the fan-out of a product vertex in a circuit is its out-degree and the fan-out of a circuit can be defined as the maximum fan-out of any node). Since fan-out is an important design factor in some applications, it will be also considered as a

Algorithm FFP_Ladner_Fischer
Input: $X = (x_1,\ x_2, \ldots, x_n)$
Output: Prefix Circuit
begin
1. generate normalized input delay profile $T_X = (t_{x_1}, \ldots, t_{x_n})$ from x_i's
2. set all the indexes live
3. **for** $j = 0 : n-2$
4. perform delay reserved shift up operation on the set of live t_{x_i}'s
   to remove isolated local minimum signals if any
5. find $(MD - SI)_j^X$ among live t_{x_i}'s
6. find $((MD - SI)_j^X + k)$
   (= first live index next to $(MD - SI)_j^X$, $k \geq 1$)
7. make index $(MD - SI)_j^X$ dead and set $t_{x_{(MD-SI)_j^X + k}} = t_{x_{(MD-SI)_j^X}} + 1$
8. **for** $m = 1 : k$
9. make a vertex with
   $x_{(MD-SI)_j^X}$ (or the newest vertex of index $(MD - SI)_j^X$) and
   $x_{(MD-SI)_j^X + m}$ (or the newest vertex of index $(MD - SI)_j^X + m$)
end

Figure 4.11: FFP-Ladner-Fischer Algorithm.

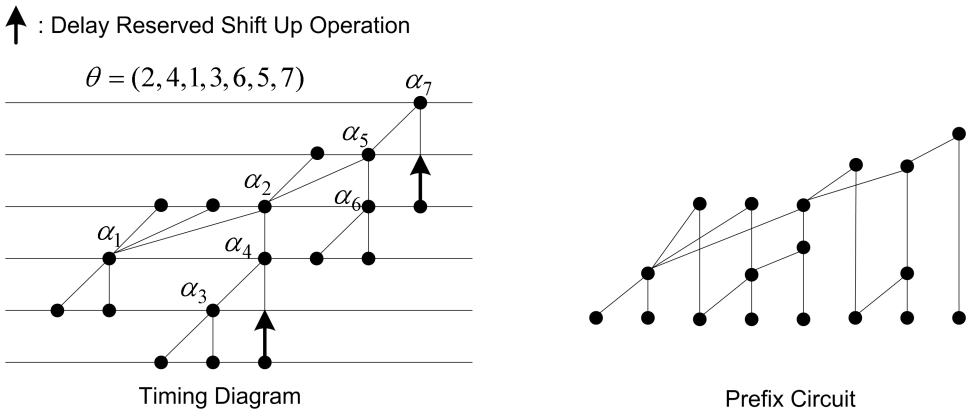

Figure 4.12: A FFP-Ladner-Fischer Algorithm example.

delay measure. In the FFP-Ladner-Fischer Algorithm shown in Figure 4.11, variable k at each iteration represents the lateral fan-out of the vertex of that iteration.

Table 4.1 summarizes test input delay profiles for the experiment of FFP-Ladner-Fischer Algorithm. There are cases of 16, 63 and 127 inputs.

$n = 16$ cases are to test effects of various input patterns on the FFP-Ladner-Fischer Algorithm. 'pos' means positive (increase in index →increase in delay) and 'neg' means negative (increase in index → decrease in delay). T_{flat16} will applied to the normal Ladner-Fischer for the reference of these cases. For more practical examples, T_{32_32} and T_{64_64} are constructed from the outputs of parallel multipliers [18]. T_{32_32} and T_{64_64} come from 32×32 and 64×64 multipliers and their numbers of inputs are 63 and 127, respectively. Therefore, for the reference of theses cases, T_{flat63}, $T_{flat127}$ are chosen and they are also applied to the normal Ladner-Fischer. $T_{32\_32}{}'$ and $T_{64\_64}{}'$ are their modified versions.

The experiment results are shown in Table 4.2. $n = 16$ examples clearly

Table 4.1: Input Delay Profiles for Experiment.

Profile ($T = (t_1, t_2, \ldots, t_n)$)
T_{flat16}=(7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7)
T_{pos16}=(0,1,2,3,4,5,6,7,7,7,7,7,7,7,7,7)
T_{neg16}=(7,7,7,7,7,7,7,7,7,6,5,4,3,2,1,0)
T_{pos_neg16}=(0,1,2,3,4,5,6,7,7,6,5,4,3,2,1,0)
T_{neg_pos16}=(7,6,5,4,3,2,1,0,0,1,2,3,4,5,6,7)
T_{wave16}=(4,5,6,7,6,5,4,5,6,7,6,5,4,5,6,7)
T_{flat63}=(11,11,11,11,11,. . .,11,11,11,11,11), ($n = 63$)
T_{32_32}=(0,0,1,2,3,4,4,5,5,6,6,7,7,7,8,8,8,8,8,9,9,9,9,9,10,10,10,10,10,10,11,11,11,11, 11,11,11,11,11,11,11,11,10,10,10,10,10,10,9,9,9,9,8,8,8,7,7,6,6,5,4,4,2)
$T_{32\_32}{}'$ =(0,0,1,2,3,4,4,5,5,6,6,7,7,7,8,8,8,8,8,9,9,9,9,9,10,10,10,10,10,10,11,11,11, 11,11,11,11,11,11,11,11,11,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10, 10,10,10)
$T_{flat127}$=(13,13,13,13,13,. . ., 13,13,13,13,13), ($n = 127$)
T_{64_64}=(0,0,1,2,3,4,4,5,5,6,6,7,7,7,8,8,8,8,8,9,9,9,9,9,10,10,10,10,10,10,11,11,11, 11,11,11,11,11,11,12,12,12,12,12,12,12,12,12,12,13,13,13,13,13,13,13,13,13,13,13, 13,13,13,14,14,14,14,14,14,14,14,14,14,14,14,14,14,13,13,13,13,13,13,13,13,13,13, 13,13,12,12,12,12,12,12,12,12,12,12,12,11,11,11,11,11,11,10,10,10,10,10,10,9,9,9, 9,8,8,8,7,7,6,6,5,4,4,2)
$T_{64\_64}{}'$=(0,0,1,2,3,4,4,5,5,6,6,7,7,7,8,8,8,8,8,9,9,9,9,9,10,10,10,10,10,10,11,11,11, 11,11,11,11,11,11,12,12,12,12,12,12,12,12,12,12,13,13,13,13,13,13,13,13,13,13,13, 13,13,13,14,14,14,14,14,14,14,14,14,14,14,14,14,14,13,13,13,13,13,13,13,13,13,13, 13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13, 13,13,13,13,13,13,13,13,13,13,13,13,13)

show how slope patterns affect the performance of prefix circuits designed by the FFP-Ladner-Fisher Algorithm. T_{pos16} shows improvement in both delay and size compared to T_{flat16} and this means that positive slope is favorable to the FFP-Ladner-Fisher Algorithm and it can reduce both size and delay. Negative slope can reduce delay but it needs a lot of size to gain speed as shown in T_{pos_neg16} and T_{neg_pos16} cases. Even worse, if the negative slope follows a long flat region, it cannot gain any speed despite a large increase in size as shown in the T_{neg16} case. T_{wave16} case shows that the FFP-Ladner-Fisher Algorithm can utilize the small fluctuating patterns well. Since T_{32_32} and T_{64_64} have positive regions, their experiment results show improvement in delay but their sizes are almost same to the normal Ladner-Fisher cases because of their negative regions.

The effect of the negative slope region may be reduced by changing part of the negative slope region into a flat slope like $T_{32\_32}{}'$ and $T_{64\_64}{}'$. Experimental results in Table 4.2 show that $T_{32\_32}{}'$ and $T_{64\_64}{}'$ have smaller size than normal Ladner-Fisher cases without increasing delay. However, if the negative region is shifted up too much, the delay of the prefix circuit will be increased eventually. Therefore, there exists an optimum amount of shift-up in the negative region which produces the minimum delay and minimum size prefix circuit. To show various usages of proposed algorithms, the iterative algorithm which finds out such optimal delay profiles was also constructed by using FFP-Ladner-Fisher as a core algorithm. The construction of the algorithm is very straightforward and its pseudo-code is omitted. The simulation results of the algorithm showed that $T_{32\_32}{}'$ and $T_{64\_64}{}'$ are actually optimal delay profiles of T_{32_32} and T_{64_64}.

Table 4.2: Results of Experiments.

Input	Size	Delay		Remarks
		Delay of FP-Tree	Lateral Fan-out	
T_{flat16}	31	11	8	from Ladner-Fisher
T_{pos16}	21	11	2	
T_{neg16}	48	11	8	
T_{pos_neg16}	43	9	8	
T_{neg_pos16}	49	9	7	
T_{wave16}	32	10	7	
T_{flat63}	162	17	31	from Ladner-Fisher
T_{32_32}	163	16	23	
$T_{32\_32}{}'$	146	16	23	
$T_{flat127}$	362	21	63	from Ladner-Fisher
T_{64_64}	371	20	38	
$T_{64\_64}{}'$	346	20	38	

4.5 Conclusion

This chapter first presents a new generalized structure and a new two step design process for prefix circuits. The main contribution of this chapter is that a minimum depth design scheme under the non-uniform input signal arrival condition is proposed using an algorithmic approach and its optimality is proved formally.

The advantage of the proposed FFP-Algorithm is that it is easy to implement and fast in run-time due to its greedy strategy and it always ensures the minimum depth prefix circuit design with the Ladner-Fischer strategy. Because the proposed algorithm provides the lower delay bound of a given input delay profile, it can also be used as a core algorithm for iterative prefix circuit design methods.

Chapter 5

Parallel Prefix Adder Design with Matrix Representation

This chapter presents a one-shot batch process that generates a wide range of designs for a group of parallel prefix adders. The prefix adders are represented by two two-dimensional matrixes and two vectors. This matrix representation makes it possible to compose two functions for gate sizing which calculate the delay and the total transistor width of the carry propagation graph of adders. After gate sizing, the critical path net list of the carry propagation graph is generated from the matrix representation for spice delay calculation. The proposed process is illustrated by generating sets of delay and total transistor width pairs for 32-bit and 64-bit cases.

5.1 Introduction

As described in Section 2.2, Knowles [5] presented complete classes of regular fan-out prefix adders which are bounded at the extremes by the Ladner-Fischer [2] and Kogge-Stone [1] graphs. In this chapter, gate level design and performance analysis of Knowles adders are examined.

In the Kogge-Stone approach, all the gates are equally sized because the fan-out of the carry generation graph is uniform. Thus, the performance analysis of the adder is quite simple. On the contrary, with the Ladner-Fischer approach, the

performance of an adder is strong function of the gate sizing because the fan-out of the adder is not uniform. Therefore, for performance analysis of the Ladner-Fischer adders, a methodology to perform gate sizing is needed.

In this chapter, a one-shot process for analyzing the complete set of Knowles prefix adders is proposed. First, a matrix representation for gate level design of the Knowles adders is proposed which is composed of two two-dimensional matrixes and two vectors. In [13], one-dimensional vectors were used to represent adders in prefix graph level but in this work, two-dimensional matrixes are used to represent adders in gate level. Second, delay and transistor width calculation functions for gate sizing are constructed from the matrixes. Because the matrix representation contains enough information to construct its original adder, it is also possible to generate a spice net list from the matrixes. These produce a one-shot batch process that generates the complete set of characteristic curves of the adders in a group.

The rest of this chapter is organized as follows. In Section 5.2, a generalized gate level layout of the carry propagation graph of the Knowles adders is proposed from observations of gate level designs of Kogge-Stone and Ladner-Fischer adders. In Section 5.3, a matrix representation for the gate level design of the Knowles adders is proposed. In Section 5.4, the matrix representation is applied to the gate sizing of the adders and Section 5.5 shows characterization results for 32-bit and the 64-bit cases. Section 5.6 concludes this chapter.

5.2 Gate Level Design of Knowles Adders

In this section, gate level designs of the Knowles adders are analyzed. For this purpose, gate level designs of the Kogge-Stone adder (Figure 5.1) and the Ladner-Fischer adder (Figure 5.2) are constructed first. In the figures, XNOR gates or equivalent logic gates before level 0 for generating $\bar{p}$ signals are omitted for simplicity. For the input gate (level 0), NAND and NOR gates are used and for the sum, XOR gates or equivalent logic gates are used. For the rest of the levels, there are four types of cells as shown in Figure 5.3. In the Ladner-Fischer adder, all the repeated inverters are removed from the carry trees to reduce the total transistor area except the output inverters of the multiple lateral fan-out gates which can reduce the total output capacitance of the gates.

The major difference between two adders is their fan-out patterns. The Kogge-Stone adder has uniform fan-out throughout all the cells, but the Ladner-Fisher adder has a nonuniform fan-out and the fan-out increases as the level increases. There are two approaches to solve the large fan-out problem. One is buffer insertion and the other is increasing gradually the size of the gates along the fan-out path (instead of increasing the size of the gate, multiple gates can be placed in parallel [9]). The problem of buffer insertion is that it adds an additional logic level to the adder which increases the overall delay of the adder. Thus, in this work, the second method is used and a frame work is presented as a solution for this gate sizing problem.

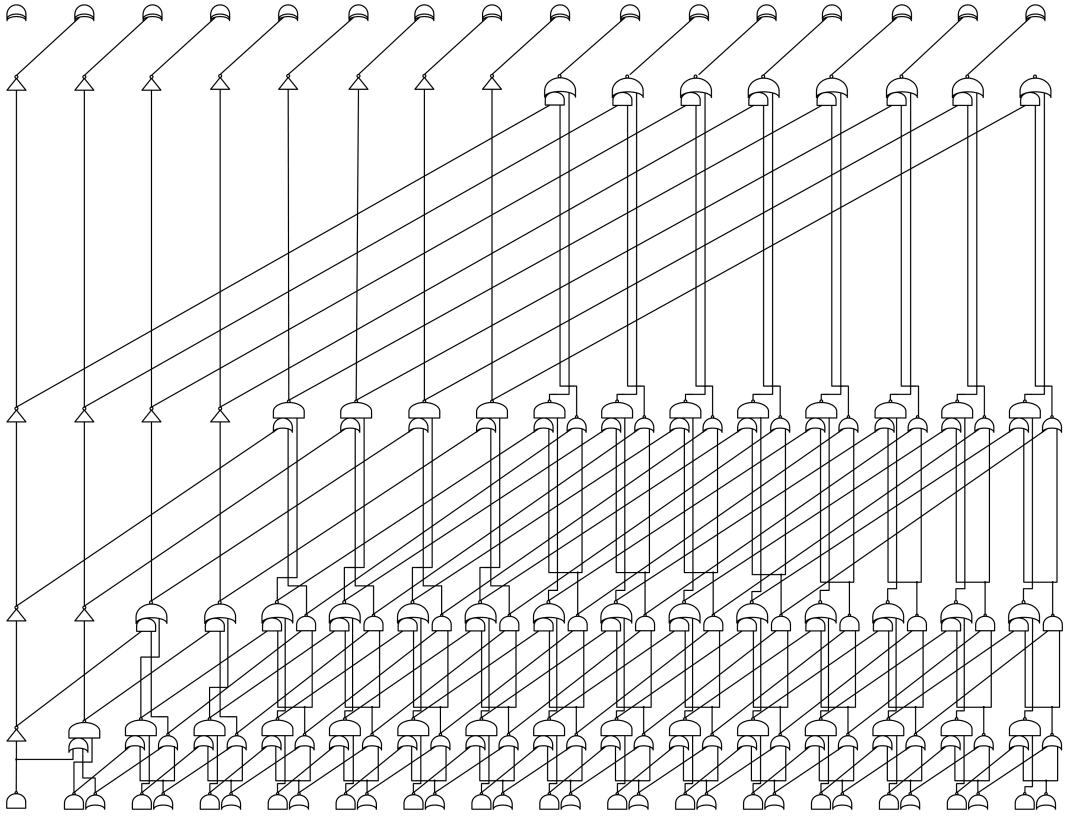

Figure 5.1: Gate level design of 16-bit Kogge-Stone adder.

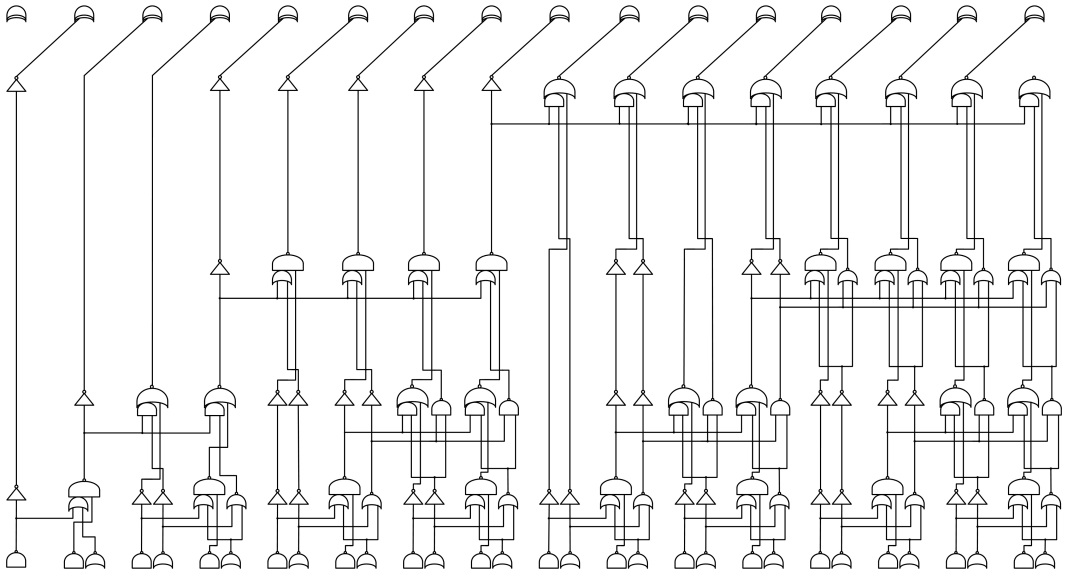

Figure 5.2: Gate level design of 16-bit Ladner-Fischer adder.

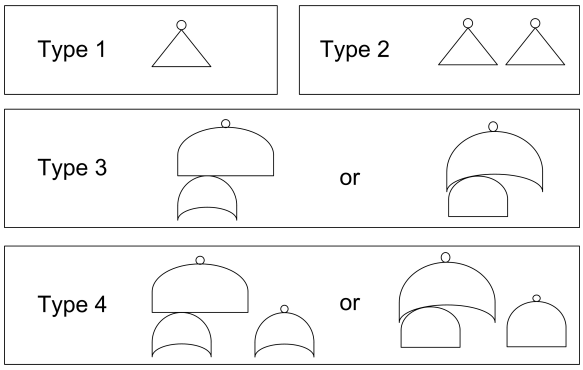

Figure 5.3: Types of cells for the carry propagation graph.

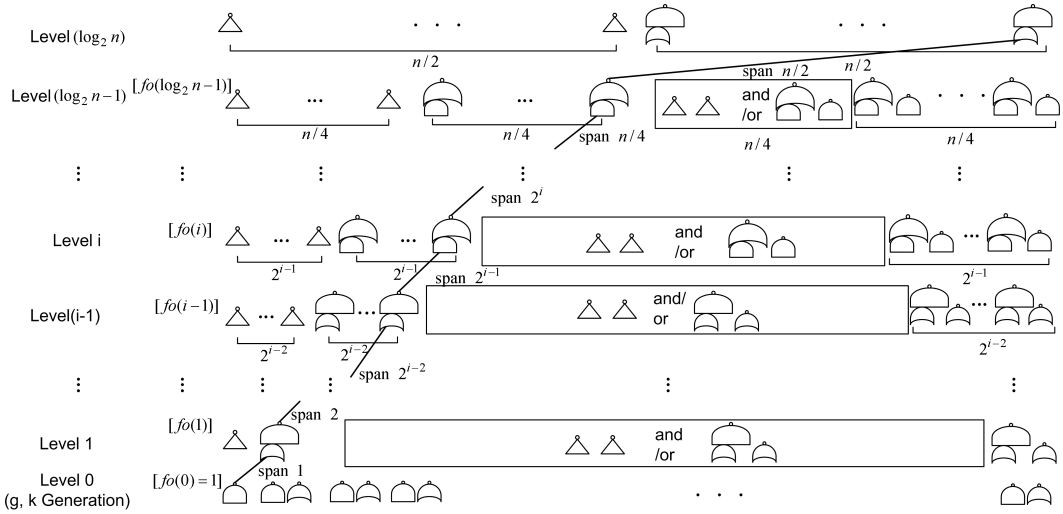

Figure 5.4: Generalized gate level layout of the carry propagation graph of the Knowles adders (fo: fan-out vector).

Through comparison of the two adders, a generalized gate level layout of the Knowles adders is presented in Figure 5.4. In the generalized layout, wire connections are not considered except for the wires for the last carry signal that is the same for all the adders in the group. In the figure, only the subsections marked by rectangles differ among adders in the group. Among the cells in the rectangular part, those which do not have any lateral input are pairs of inverters (type 2) and the others are type 4. This can be determined by the fan-out vector of the adder. If the fan-out of the level i is 2^i, it blocks the lateral connection of the cells in the lower level like the Ladner-Fischer adder. In the i^{th} level, first 2^{i-1} cells are type 1, cells from $2^{i-1}+1$ to 2^i are type 3, cells from $n-2^{i-1}+1$ to n are type 4 and the rest of the cells are of type 2 or type 4.

5.3 Matrix Representation of Knowles Adders

In this section, a matrix representation of the Knowles adders is proposed. The matrix representation is composed of two two-dimensional matrixes, the type matrix (T) and the size matrix (S), and two vectors, the fan-out vector (fo) and the size vector (s). The two dimensional representation is very natural for the prefix adders because a two dimensional index can represent a cell of the adders. The T matrix represents the types of the cells among four types shown in Figure 5.3. The S matrix marks cells which have variable size. Because level 0 is fixed for all the adders with the same operand width, the index of the matrixes starts from 1. To match the row index of the matrixes with the graphical representation of the adders in this dissertation, the row index starts from bottom. These two dimensional matrixes are constructed from a given fan-out vector fo using the analysis results in Section 5.2.

The size vector s represents the multiplication factors for the variable size cells as follows:

$$s = (s(1), \ldots, s(\log_2 n - 1)),\ \textit{where } n = \textit{operand width}. \tag{5.1}$$

All the gates in the last level are assumed to be of minimum size and therefore $s(\log_2 n)$ is excluded from the size vector. If they become non-minimum for any reason, the size of the other gates is simply increased at the same rate as the last level.

Algorithm T_matrix
Input: $fo = (fo(0), \ldots, fo(\log_2 n - 1))$
Output: $T(i, j)$ matrix
begin
1. set $T(i, j) = 4$ for all $i = 1, 2, \ldots, \log_2 n$ and $j = 1, 2, \ldots, n$
2. **for** $i = 1 : \log_2 n$
3. set $T(i, j) = 1$ for $j = 1, 2, 3, \ldots, 2^{i-1}$
4. set $T(i, j) = 3$ for $j = 2^{i-1} + 1, \ldots, 2^i$
5. **for** $k = (2^i / fo(i-1)) + 1 : (n - 2^{i-1})/fo(i-1)$
6. **for** $m = i : \log_2 n - 1$
7. **if**$\{(fo(m) == 2^m) \wedge (\lceil k \cdot fo(i-1)/2^m \rceil \neq \lceil (k \cdot fo(i-1) - 2^{i-1})/2^m \rceil)\}$
set $T(i, fo(i-1) \cdot (k-1) + j) = 2$ for $j = 1, 2, \ldots, fo(i-1)$
end

Figure 5.5: Algorithm for generating T matrix.

5.3.1 T matrix

The value of $T(i, j)$ represents the type of the cell in Figure 5.3 as follows:

$$\text{Type k} : T(i, j) = k, \ \textit{where } k = 1, 2, 3, 4.$$

Figure 5.5 is an algorithm for constructing the T matrix. As explained Section 5.2, in the i^{th} level, the first 2^{i-1} cells are set to type 1, cells from $2^{i-1} + 1$ to 2^i are set to type 3, cells from $n - 2^{i-1} + 1$ to n are set to type 4 and the rest of the cells are checked by the innermost if-statement (line 7 in Figure 5.5) which sets a cell to type 2 if the cell doesn't have any lateral input.

5.3.2 S matrix

There is one assumption for the S matrix to reduce the number of variables and make the gate sizing problem manageable:

All the variable size cells in the same row have the same multiplication factor for their sizes.

Because the multiplication factors ($s_i \in s$) will be calculated along the critical path, this assumption does not affect the delay estimation but it may overestimate the total transistor width because the sizes of the gates that are off the critical path may be reduced by a finer gate sizing process. However, the error due to the assumption is not significant because near the Ladner-Fisher end, the number of variable size cells is small and near the Kogge-Stone end, the multiplication factors are not big because fan-outs of gates are small and regular.

The convention for the values of $S(i, j)$ is:

$$\begin{aligned} \text{Empty cell} &: S(i,j) = -1 \\ \text{Minimum size cell} &: S(i,j) = 0 \\ \text{Variable size cell} &: S(i,j) = i \end{aligned}$$

Figure 5.6 is an algorithm for constructing the S matrix. The algorithm for the S matrix is composed of two steps. In the first step, it finds variable size cells. It searches the rows of the T matrix from the top to the bottom. During the iteration, it first finds multiple lateral fan-out cells (lines 3-4 and 6-7 in Figure 5.6) and after that, it also finds the cells connected with variable size cells of upper level (lines 8-11 in Figure 5.6) because variable size cells may need large drivers.

In the second step, it removes unnecessary repeated inverters except for output inverters of lateral fan-out gates which can reduce the total output capacitance of the gates.

Algorithm S_matrix
Input: T, $fo = (fo(0), \ldots, fo(\log_2 n - 1))$
Output: $S(i, j)$ matrix
begin
1. set $S(i, j) = 0$ for all $i = 1, 2, \ldots, \log_2 n$ and $j = 1, 2, \ldots, n$
2. set $i = \log_2 n$
3. **for** $k = (2^{i-1}/fo(i-1)) + 1 : (n/fo(i-1))$
4. **if** $(fo(i-1) > 1)$
$S(i-1,\ fo(i-1) \cdot k - 2^{i-1}) = i - 1$
5. **for** $i = \log_2 n - 1 : 2 : -1$
6. **for** $k = (2^{i-1}/fo(i-1)) + 1 : (n/fo(i-1))$
7. **if** $\{fo(i-1) > 1 \wedge (T(i, fo(i-1) \cdot k) == 3\ or\ 4)\}$
$S(i-1,\ fo(i-1) \cdot k - 2^{i-1}) = i - 1$
8. **for** $j = 1 : n$
9. **if** $(S(i, j) == i)$
10. $S(i-1, j) = i - 1$
11. **if** $(j - 2^{i-1} > 0)$
$S(i-1, j - 2^{i-1}) = i - 1$
12. **Remove repeated inverters**
end

Figure 5.6: Algorithm for generating S matrix.

Figure 5.7 shows examples of T and S matrixes. In the figure, the index of bottom row is 1 to match the row index of the matrixes with the graphical representation of the adders.

In summary, the proposed matrix representation of the Knowles adders is a mathematical representation of a gate level design of the adders. The T matrix defines the types of the gates, the S matrix and the s vector define the size of the gates and the fo vector defines how to connect the gates.

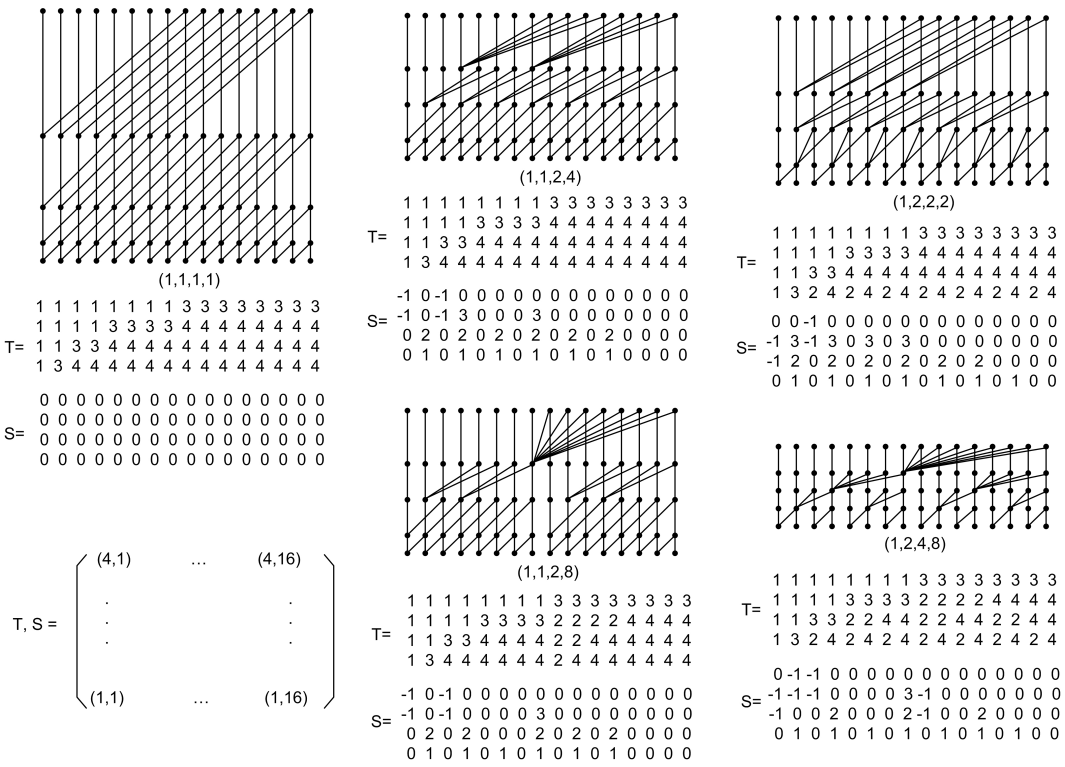

Figure 5.7: T and S matrix examples for the 16-bit case.

5.4 Characterization of Knowles Adders

One of the advantages of the matrix representation of the Knowles adders is that it can be used to make functions for adder characterization. In the matrix representation, T, S and fo are constant but the size vector s is variable according to a design choice. Thus, s will be the input variable for the functions. In this work, two functions, one for delay and the other for total transistor width, will be constructed. The total transistor width is selected as a performance measure because both the dynamic power and the static power are directly dependant on it [22]. The functions are used for gate sizing through mathematical optimization.

Another advantage of the matrix representation is that a spice net list can easily be generated from it because it is a mathematical equivalent of the gate level design of the adders. Thus, a batch process generating the complete set of characteristic curves of the adders is proposed using 6 characterization points on the delay and transistor width plane.

5.4.1 Functions for Gate Sizing

The delay model using logical and electrical efforts [8] is simple and many previous works [9–12] have used it for the delay estimation or both gate sizing and delay estimation. In this work, it is only used for gate sizing because the adder delay will be more accurately calculated through spice simulation during the proposed batch characterization process.

The delay equation for the effort model is

$$d = \tau(gh + p). \tag{5.2}$$

(where τ is a constant for unit conversion, g is the logical effort, h is the electrical effort and p is the parasitic delay of the gate). In the ideal case, g and p are independent of the size of the gate, and the only factor that is affected by gate sizing is the electrical effort h which is the ratio of the capacitive load driven by the gate to the input capacitance at the corresponding input pin. However, simulation results show that g and p also vary with the input transition time [12] and the size of the gate and as a result of this, the delay estimation with the effort model is inaccurate. Nevertheless, the effort model can be used for gate size optimization because approximate optimal size vectors are enough for constructing characteristic curves.

Since g mainly depends on the topology of the gate, it is almost independent of the size of the gate and therefore,

$$\left|\frac{\partial g}{\partial h}\right| \ll 1. \tag{5.3}$$

The parasitic delay p occurs primary due to the capacitance of the source/drain regions which depends on the layout geometry. Thus, p is also almost independent of the size of the gate and

$$\left|\frac{\partial p}{\partial h}\right| \ll 1. \tag{5.4}$$

By Equations (5.3) and (5.4),

$$\frac{\partial d}{\partial h} = \tau\left(\frac{\partial g}{\partial h}h + g + \frac{\partial p}{\partial h}\right) \approx \tau g. \tag{5.5}$$

Because the rate of change is the main factor for optimization process, a reduced version of Equation (5.2) is proposed for the gate sizing purpose by Equation (5.5):

$$d = gh. \tag{5.6}$$

In Equation (5.6), τ is also omitted because τ is just a constant for unit conversion. When the optimization cost is the delay only, dropping the constant term does not have any effect on the solution, but when the optimization cost is the multiplication of delay and another cost, it makes the gate sizing tend to minimize the delay more.

The critical path of the carry generation graphs of Knowles adders is NOR$\rightarrow$ OAI $\rightarrow$ AOI $\rightarrow \cdots \rightarrow$ XOR from the figures in Section 5.2. Because OAI and AOI alternate in the critical path and their characteristics are almost the same, an average characteristic value is used for them for simplicity. The average value is denoted by 'AOAI'. Cin represents the input capacitance of the minimum size gate. $size_{in}$ is the size multiplication factor of the input gates which is introduced to control the overall delay of the carry propagation graph when the delay constraint is not satisfied with minimum size gates.

Figure 5.8 shows a pseudo code for the delay calculation function of the carry propagation graph. In the pseudo code, h's of lines 2-3, 6-7 and 10-11 are decided by the fo of the next level by the property that, if the fan-out of the level i is 2^i, it blocks the lateral connection of the cells in the lower level like the Ladner-Fischer adder. The delay of the last level (level $\log_2 n$) is excluded because it is independent of the size vector.

Function **Delay**
Input: $s = (s(1), \ldots, s(\log_2 n - 1))$
Output: D (delay from level 0 to level $(\log_2 n - 1)$)
Parameter: $fo,\ g_{nor},\ g_{aoai},\ Cin_{inv},\ Cin_{nor},\ Cin_{aoai},\ size_{in}$
begin

1. $i = 0$
2. **if** $(fo(i+1) == 2^{i+1})$
   $h = \{(Cin_{aoai} + Cin_{nor}) \cdot s(i+1)\}/(Cin_{nor} \cdot size_{in})$
3. **else**
   $h = \{(Cin_{aoai} + Cin_{nor}) \cdot s(i+1) + Cin_{nor}\}/(Cin_{nor} \cdot size_{in})$
4. $D = g_{nor} \cdot h$
5. **for** $i = 1 : \log_2 n - 3$
6. **if** $(fo(i+1) == 2^{i+1})$
   $h = \{s(i+1) + fo(i) - 1 + Cin_{inv}/Cin_{aoai}\}/s(i)$
7. **else**
   $h = \{2 \cdot s(i+1) + fo(i) - 1\}/s(i)$
8. $D = D + g_{aoai} \cdot h$
9. $i = \log_2 n - 2$
10. **if** $(fo(i+1) == 2^{i+1})$
    $h = \{s(i+1) + fo(i) - 1 + Cin_{inv}/Cin_{aoai}\}/s(i)$
11. **else**
    $h = \{s(i+1) + fo(i) - 1 + s(i+1) \cdot Cin_{inv}/Cin_{aoai}\}/s(i)$
12. $D = D + g_{aoai} \cdot h$
13. $i = \log_2 n - 1$
14. $h = \{fo(i) + Cin_{inv}/Cin_{aoai}\}/s(i)$
15. $D = D + g_{aoai} \cdot h$

end

Figure 5.8: Function for delay calculation.

Making a function for calculating the total transistor width of the carry propagation graph is quite straightforward compared to the delay function. w represents the total transistor width of the minimum size gate. Because NOR and NAND gates alternate in the graph, average value of their total transistor width is used for simplicity. 'NDNR' denotes the average value.

Figure 5.9 shows a pseudo code for the total transistor width calculation function of the carry propagation graph. Unlike the delay function, this includes both size vector dependant terms and size vector independent terms and represents the real total transistor width of the carry propagation graph.

Function **Width**
Input: $s = (s(1), \ldots, s(\log_2 n - 1))$
Output: W (total transistor width of carry propagation tree)
Parameter: $fo,\ T,\ S,\ w_{inv},\ w_{aoai},\ w_{ndnr},\ size_{in}$
begin
1. $s = (s(1), \ldots, s(\log_2 n - 1), 1)$
2. $w = (w_{inv}, 2 \cdot w_{inv}, w_{aoai}, w_{aoai} + w_{ndnr})$
3. $W = w_{ndnr} \cdot (2n - 1) \cdot size_{in}$
4. **for** $i = 1 : \log_2 n$
5. **for** $j = 1 : n$
6. **switch** $S(i, j)$
case -1: %do nothing %
case 0: $W = W + w(T(i, j))$
otherwise : $W = W + s(i) \cdot w(T(i, j))$
end

Figure 5.9: Function for transistor width calculation.

5.4.2 Characteristic Curve

Using the proposed delay and transistor width functions, optimal gate sizing is performed with two costs, delay only and delay-width product. The gate sizing is performed by selecting the size vector which minimizes a given cost using any appropriate algorithm.

Because all the constant terms are excluded from the delay function, the delay-width product cost tends to reduce the delay more than the total transistor width. This problem can be solved by inserting an intermediate point between the minimum size point and the minimum delay-width product point. To refine the characteristic graph, two additional intermediate points are inserted between the minimum delay-width product point and the minimum delay point. Thus, six characteristic points are proposed as follows.

- Minimum size
- Intermediate point 1
- Minimum Delay-Width product
- Intermediate point 2
- Intermediate point 3
- Minimum delay

After gate sizing, spice simulation is performed to obtain a realistic delay for the carry propagation graph. Because the proposed matrix representation contains enough information for gate level implementation of the adders, the critical path

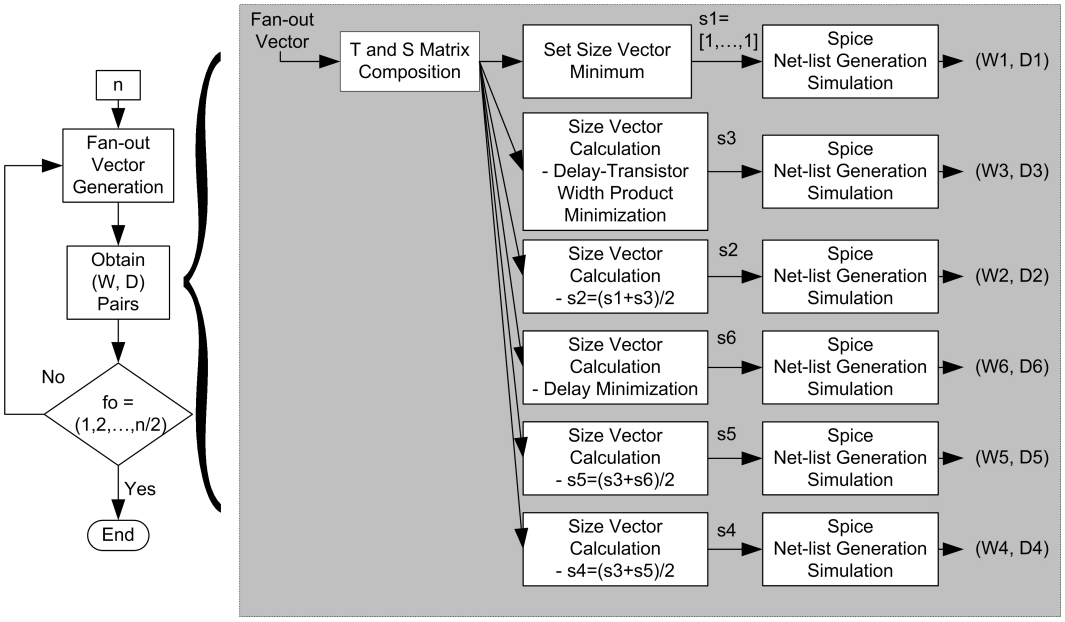

Figure 5.10: Block diagram for the proposed batch characterizing process.

net list of the graph can easily be generated from it. Unlike the delay function, the delay of the last level (level $\log_2 n$) is included during the spice simulation to obtain the whole delay of the carry propagation graph.

Figure 5.10 shows the proposed process. In the figure, Fan-out vectors are generated from $(1, 1, \ldots, 1)$ to $(1, 2, \ldots, n/2)$ and six pairs of (W, D) are generated at each iteration. W is calculated from the total transistor width generating function and D is calculated with spice simulation.

5.5 Simulation

The proposed batch process is applied to the TSMC 0.18μ technology [23] whose minimum width (w_{min}) is 0.27μ. For simplicity, the sizes of all transistors are rounded to multiples of w_{min} and transistor width is expressed in number of w_{min}. All the spice simulations are performed at room temperature and a 1.8V power supply voltage.

For the first step of the proposed process, static library cells are composed. The transistor width of a gate is determined to have rise and fall times that are as close to equal as possible by spice simulation. The results are shown in Figure 5.11.

The effort model of each library cell is simulated under the conditions that the fan-out of the input signal is 3 (the input line is connected to two additional dummy gates together with the target gate) and the size of the gate is 3. For all the gates, the worst case input pins (the furthest from the output node) are selected for

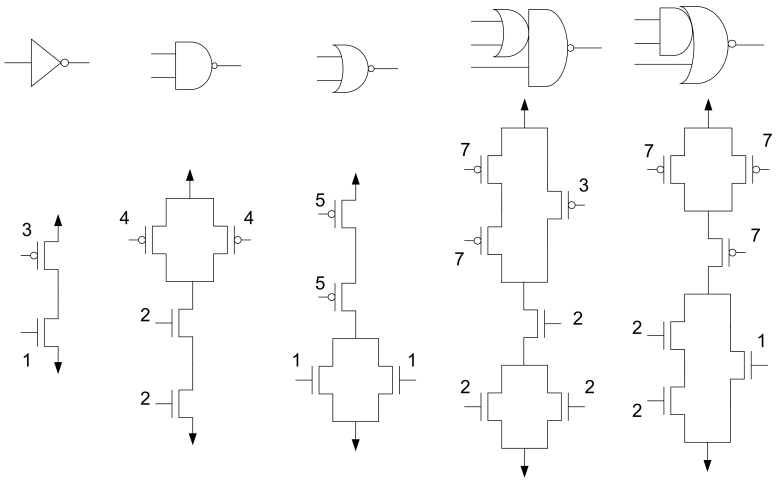

Figure 5.11: Transistor level design of library Cells ($L = 0.18\mu, W = 0.27\mu \times$ *multiplication number of the transistor*) .

Table 5.1: The Effort Models of the Library Cells.

	Inv	Nand	Nor	OAI	AOI
$\tau g[ps]$	14.56	17.61	22.88	26.50	26.33
$\tau p[ps]$	34.83	44.11	59.46	65.97	67.75

Table 5.2: Parameters for the Proposed Batch Process.

	Parameter	Value
Delay Function	g_{nor}	22.88
	g_{aoai}	(26.50+26.33)/2=26.415
	Cin_{inv}	4
	Cin_{nor}	6
	Cin_{aoai}	(9+9)/2=9
$size_{in}$		1
Width Function [number of w_{min}]	w_{inv}	4
	w_{ndnr}	(12+12)/2 = 12
	w_{aoai}	(23+26)/2 = 24.5

the simulation. τg and τp are calculated using least square method from the data which is generated by three h's, 1, 3 and 6. The results are shown in Table 5.1.

From Figure 5.11 and Table 5.1, all parameters of the proposed batch process except for $size_{in}$ are decided as shown in Table 5.2. $size_{in}$ is selected as 1 in this simulation but it can be used as a tuning parameter which controls the ratio of the adder delay to the overall delay of the total critical path of a given design.

The set of parameters in Table 5.2 is applied to 32-bit case and the results are shown in Table 5.3. The output capacitance of the gates of the last level (level $\log_2 n$) for spice simulation is assumed to be $1.5 \times Cin_{aoai}$. Since the delay function uses

an approximate model, $D4$ or $D5$ is the minimum in 16 cases (the bold numbers in Table 5.3) out of 42, but the difference is small because the delay is already saturated after $D4$ and the delay is not reduced much more by increasing the size of the gates. The approximate delay model provides an upper bound for the size vector and the actual minimum delay size vector can be located using the intermediate points of the process.

The 64-bit case has also been simulated and Figures 5.12 and 5.13 show the W-D curves of some Knowles carry propagation graphs from 32-bit case and 64-bit case, respectively. As expected, Kogge-Stone $(1, 1, \ldots, 1)$ has the best delay, but the delay advantage will be decreased if wire delay is included. If both fast delay and moderate wire load are needed, $(1, 2, \ldots, 2)$ is a suitable choice. On each Figure, there is a crossing level (around $830ps$ for Figure 5.12 and around $1000ps$ for Figure 5.13). Above the crossing level, Ladner-Fischer $(1, 2, \ldots, n/2)$ is the best for both delay and area because Ladner-Fischer needs the least wire among all the adders in the same group. Below the crossing level, $(1, 2, 4, \ldots, 4)$ and $(1, 2, 4, 8, \ldots, 8)$ show better performance, but the performance gap is decreased if wire delay and area are considered. The figures show clearly which adder is the best for a given design criteria when only minimum size gates are available.

If adequate layout information is available, wire delay can be inserted into the delay function [15] and the spice net list to make the proposed process able to deal with the effects of wire delay. In this case, $S(i, j)$ of Kogge-Stone adder which has lateral fan-out will be also changed from 0 to i to mark it as a variable size cell to drive wire load.

Table 5.3: Simulation Result of 32-bit Case ($W[number\ of\ w_{min}], D[ps]$).

Fan-out	Min. Size	Inter. 1	Opt. Pro.	Inter. 2	Inter. 3	Min. Delay
	s2	s3	s4	s5	s6	
	(W1,D1)	(W2,D2)	(W3,D3)	(W4,D4)	(W5,D5)	(W6,D6)
(1,1,1,1,1)		(1,1,1,1)	(1,1,1,1)	(1,1,1,1)	(1,1,1,1)	(1,1,1,1)
	5378,649	5378,649	5378,649	5378,649	5378,649	5378,649
(1,1,1,1,2)		(1,1,1,1)	(1,1,1,1)	(1,1,1,1)	(1,1,1,1)	(1,1,1,1)
	5378,680	5378,680	5378,680	5378,680	5378,680	5378,680
(1,1,1,1,4)		(1,1,1,2)	(1,1,1,2)	(1,1,1,2)	(1,1,1,2)	(1,1,1,2)
	5378,740	5435,725	5435,725	5435,725	5435,725	5435,725
(1,1,1,1,8)		(1,1,1,2)	(1,1,1,2)	(1,1,1,2)	(1,1,2,3)	(1,1,2,3)
	5378,858	5406,789	5406,789	5406,789	5497,781	5497,781
(1,1,1,1,16)		(1,1,1,3)	(1,1,2,5)	(1,1,2,5)	(1,2,3,6)	(1,2,3,7)
	4935,1091	4984,847	5095,814	5095,**814**	5319,**822**	5343,**820**
(1,1,1,2,2)		(1,1,1,1)	(1,1,1,1)	(1,1,1,1)	(1,1,1,1)	(1,1,1,1)
	5346,711	5346,711	5346,711	5346,711	5346,711	5346,711
(1,1,1,2,4)		(1,1,1,2)	(1,1,1,2)	(1,1,1,2)	(1,1,1,2)	(1,1,1,2)
	5346,771	5403,756	5403,756	5403,756	5403,756	5403,756
(1,1,1,2,8)		(1,1,1,2)	(1,1,1,2)	(1,1,1,2)	(1,1,2,3)	(1,2,3,4)
	5346,890	5374,821	5374,821	5374,821	5760,**797**	6549,**807**
(1,1,1,2,16)		(1,1,2,3)	(1,1,2,5)	(1,1,2,5)	(1,2,3,6)	(1,2,3,7)
	4903,1123	5138,866	5187,831	5187,**831**	5733,**833**	5758,831
(1,1,1,4,4)		(1,1,2,2)	(1,1,2,2)	(1,1,2,2)	(1,2,3,3)	(1,2,3,3)
	5330,836	5565,784	5565,784	5565,**784**	6013,790	6013,**790**
(1,1,1,4,8)		(1,1,2,2)	(1,1,2,3)	(1,1,2,3)	(1,2,3,4)	(1,2,3,4)
	5330,956	5537,849	5565,831	5565,831	5984,829	5984,829
(1,1,1,4,16)		(1,1,2,3)	(1,1,2,5)	(1,1,2,6)	(1,2,3,7)	(2,3,4,8)
	4887,1188	5048,901	5097,864	5121,862	5469,**854**	6254,**860**
(1,1,1,8,8)		(1,1,2,3)	(1,1,3,4)	(1,1,4,4)	(1,2,5,5)	(1,2,6,6)
	4702,1086	4859,873	4987,838	5086,835	5426,825	5554,824
(1,1,1,8,16)		(1,1,2,4)	(1,1,3,6)	(1,1,4,7)	(1,2,5,8)	(1,2,6,10)
	4466,1320	4639,918	4788,866	4912,856	5248,844	5396,841
(1,1,2,2,2)		(1,1,1,1)	(1,1,1,1)	(1,1,1,1)	(1,2,2,2)	(1,2,2,2)
	5346,743	5346,743	5346,743	5346,**743**	6295,750	6295,**750**
(1,1,2,2,4)		(1,1,1,2)	(1,1,1,2)	(1,1,1,2)	(1,2,2,3)	(1,2,2,3)
	5346,804	5403,788	5403,788	5403,788	6295,785	6295,785
(1,1,2,2,8)		(1,1,1,2)	(1,1,1,2)	(1,1,1,2)	(1,2,2,3)	(2,3,3,4)
	5346,923	5374,853	5374,853	5374,853	6238,**829**	7614,**835**
(1,1,2,2,16)		(1,1,2,3)	(1,1,2,5)	(1,1,2,6)	(1,2,3,7)	(2,3,4,8)
	4871,1155	5106,898	5155,863	5179,861	5838,**849**	7009,**855**
(1,1,2,4,4)		(1,1,2,2)	(1,1,2,2)	(1,1,2,2)	(1,2,3,3)	(2,3,3,3)
	5330,869	5565,816	5565,816	5565,816	6279,**807**	7270,**818**
(1,1,2,4,8)		(1,1,2,2)	(1,1,2,3)	(1,1,2,3)	(1,2,3,4)	(2,3,4,5)
	5330,989	5537,881	5565,863	5565,863	6251,**846**	7448,**851**

(1,1,2,4,16)		(1,1,2,3)	(1,1,2,5)	(1,1,2,6)	(1,2,3,7)	(2,3,4,8)
	4855,1221	5016,933	5065,896	5089,894	5637,**871**	6696,**872**
(1,1,2,8,8)		(1,1,2,3)	(1,1,3,4)	(1,1,4,4)	(1,2,5,5)	(2,3,7,6)
	4702,1118	4859,906	4987,870	5086,866	5492,**841**	6509,**843**
(1,1,2,8,16)		(1,1,2,4)	(1,1,3,6)	(1,1,4,7)	(1,2,5,9)	(2,3,7,11)
	4434,1352	4607,950	4756,898	4880,888	5306,860	6343,858
(1,1,4,4,4)		(1,2,2,2)	(1,2,2,2)	(1,2,2,2)	(1,3,3,3)	(1,3,3,3)
	4710,921	5195,807	5195,807	5195,807	5680,787	5680,787
(1,1,4,4,8)		(1,2,2,2)	(1,2,2,3)	(1,2,3,4)	(1,3,4,4)	(2,5,5,5)
	4710,1041	5167,872	5195,854	5402,837	5830,**820**	7048,**821**
(1,1,4,4,16)		(1,2,2,4)	(1,2,3,6)	(1,3,4,7)	(2,4,5,8)	(2,5,6,10)
	4308,1274	4743,897	4903,864	5289,842	6186,842	6596,837
(1,1,4,8,8)		(1,2,3,3)	(1,2,4,4)	(1,2,5,5)	(1,3,7,6)	(2,5,9,7)
	4348,1172	4854,864	4982,840	5110,831	5587,811	6825,810
(1,1,4,8,16)		(1,2,3,4)	(1,2,4,7)	(1,2,5,8)	(1,3,7,10)	(2,5,10,13)
	4064,1405	4586,908	4759,861	4883,851	5381,826	6764,823
(1,2,2,2,2)		(1,1,1,1)	(1,1,1,1)	(1,1,1,1)	(2,2,2,2)	(2,2,2,2)
	4895,749	4895,749	4895,749	4895,**749**	6394,750	6394,**750**
(1,2,2,2,4)		(1,1,1,2)	(1,1,1,2)	(1,1,1,2)	(2,2,2,3)	(2,3,3,3)
	4895,810	4952,794	4952,794	4952,794	6394,**785**	7230,**786**
(1,2,2,2,8)		(1,1,1,2)	(1,1,1,2)	(1,1,1,2)	(2,2,2,3)	(2,3,3,4)
	4895,928	4924,859	4924,859	4924,859	6337,829	7201,825
(1,2,2,2,16)		(1,1,2,3)	(1,1,2,5)	(1,1,2,6)	(2,2,3,7)	(2,3,4,8)
	4450,1161	4685,904	4734,869	4758,867	5967,848	6626,845
(1,2,2,4,4)		(1,1,2,2)	(1,1,2,2)	(1,1,2,2)	(2,2,3,3)	(2,3,3,3)
	4879,875	5115,823	5115,823	5115,823	6378,**807**	6857,**808**
(1,2,2,4,8)		(1,1,2,2)	(1,1,2,3)	(1,1,2,3)	(2,2,3,4)	(2,3,4,5)
	4879,995	5086,887	5115,870	5115,870	6350,846	7035,841
(1,2,2,4,16)		(1,1,2,3)	(1,1,2,5)	(1,1,2,6)	(2,2,3,7)	(2,3,4,8)
	4434,1227	4595,939	4644,903	4668,901	5765,871	6313,862
(1,2,2,8,8)		(1,1,2,3)	(1,1,3,4)	(1,1,4,4)	(2,2,5,5)	(2,3,7,6)
	4292,1124	4449,912	4577,877	4676,873	5631,841	6136,833
(1,2,2,8,16)		(1,1,2,4)	(1,1,3,6)	(1,1,4,7)	(2,2,5,9)	(2,3,7,11)
	4040,1358	4213,956	4362,904	4486,894	5461,859	5987,848
(1,2,4,4,4)		(1,2,2,2)	(1,2,2,2)	(1,3,3,3)	(2,4,4,3)	(2,5,5,4)
	4466,928	4951,812	4951,812	5436,791	6414,782	6899,780
(1,2,4,4,8)		(1,2,2,2)	(1,2,2,3)	(1,3,3,4)	(2,4,4,4)	(2,5,5,5)
	4466,1047	4923,877	4951,859	5408,830	6385,817	6842,811
(1,2,4,4,16)		(1,2,2,4)	(1,2,3,6)	(1,3,4,7)	(2,4,5,8)	(2,5,6,10)
	4064,1280	4499,902	4659,869	5045,846	5980,833	6390,827
(1,2,4,8,8)		(1,2,3,3)	(1,2,4,4)	(1,3,5,5)	(2,4,7,6)	(2,5,9,7)
	4056,1178	4562,869	4690,845	5067,822	6094,805	6571,800
(1,2,4,8,16)		(1,2,3,4)	(1,2,4,7)	(1,3,5,8)	(2,4,7,10)	(2,5,10,13)
	3788,1411	4310,913	4483,866	4857,842	5904,821	6525,813

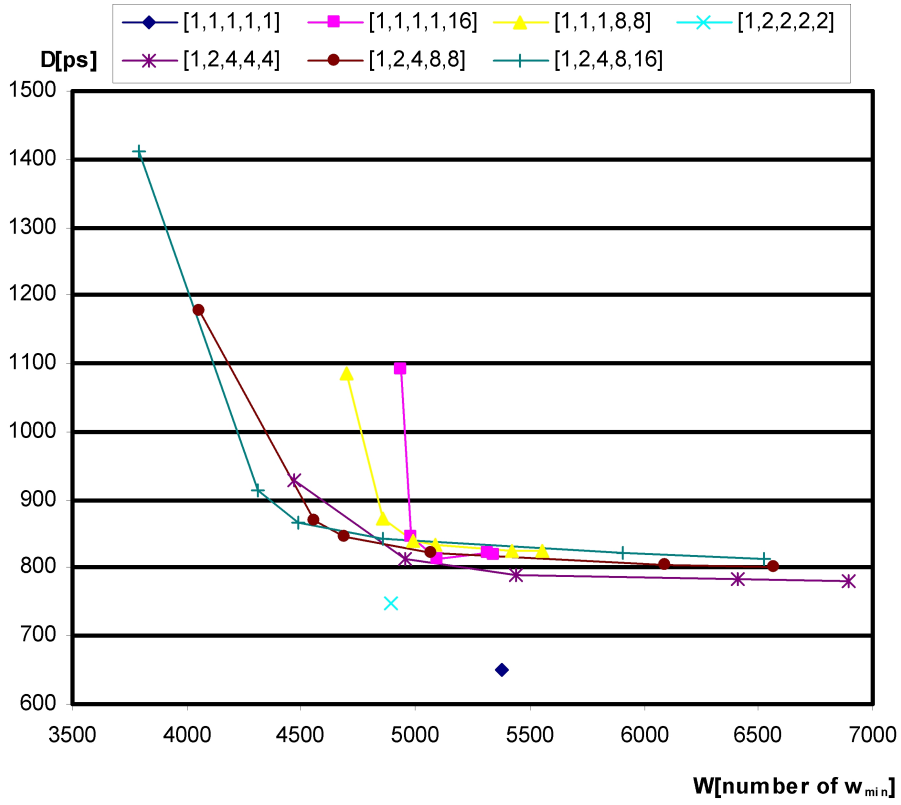

Figure 5.12: W-D Graphs for the selected elements of Table 5.3.

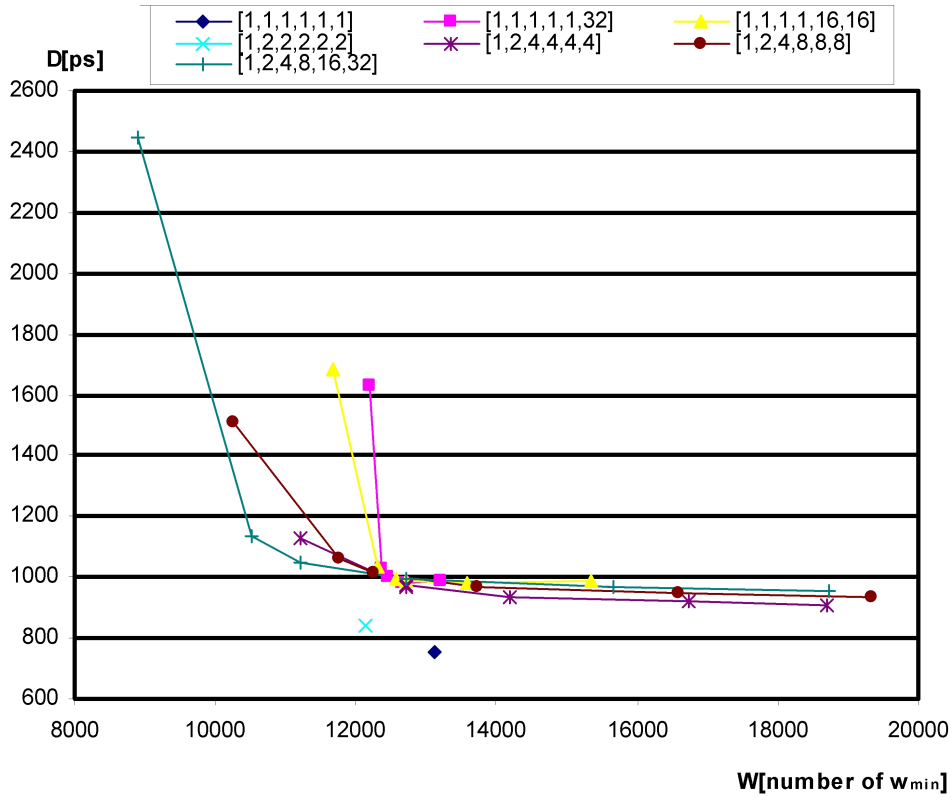

Figure 5.13: W-D Graphs for the selected elements of 64-bit case.

5.6 Conclusion

A matrix representation for the gate level design of Knowles adders is proposed which is composed of two two-dimensional matrixes and two vectors. The matrix representation is successfully applied to gate sizing of the adders.

A fast characterization process for Knowles adders is proposed which can be used for selecting an adder during the early design phase. It is easily programmable through CAD tools such as MATLAB. It is a one-shot process which generates all characteristic curves of a group of adders automatically. For accurate delay estimation, the process uses spice simulation. The set of characteristic curves provides a designer with useful information such as lower bounds for the delay, the total transistor width and the W-D slope.

Chapter 6

Conclusion

In this dissertation, two innovations on parallel prefix circuit and adder design have been presented:

- a minimum depth prefix circuit design scheme under the non-uniform input signal arrival condition

- a one-shot process analyzing characteristics of the complete set of Knowles prefix adders.

Because they are presented by algorithms or pseudo-codes, they can easily be implemented with or integrated into CAD tools.

Chapter 4 presents a minimum depth prefix circuit design scheme under the non-uniform input signal arrival condition. In the chapter, first, a new generalized structure and a new two step design process for prefix circuits are proposed. Based on these, FFP-Problem is formulated. For a solution for the FFP-Problem, FFP-Algorithm is proposed and its optimality is proved formally. The advantage of the proposed FFP-Algorithm is that it is easy to implement and fast in run-time due to its greedy strategy and it always ensures the minimum depth prefix circuit design with the Ladner-Fischer strategy. To complete the proposed two step design process, FFP-Ladner-Fischer algorithm is proposed and its usefulness is verified by

examples. Because the proposed FFP-Ladner-Fisher algorithm provides the lower delay bound of a given input delay profile, it can also be used as a core algorithm for iterative prefix circuit design methods.

In chapter 5, a one-shot process analyzing characteristics of the complete set of Knowles prefix adders is proposed. First, from observations of gate level designs of Kogge-Stone and Ladner-Fischer adders, a generalized gate level layout of the carry propagate graph of the Knowles adders is proposed. Based on this, a matrix representation for the gate level design of the Knowles adders is proposed which is composed of two two-dimensional matrixes and two vectors. The matrix representation is successfully applied to gate sizing of the adders by composing two functions, delay function and total transistor width function, from it. Through the matrixes and the functions, a fast characterization process for the group of the Knowles adders is proposed which can be used for selecting an adder during the early design phase. It is easily programmable through CAD tools such as MATLAB. It is a one-shot process which generates all the characteristic curves of a group of Knowles adders automatically. For accurate delay estimation, the process uses spice simulation. The set of characteristic curves provide a designer with useful information such as lower bounds for the delay and the total transistor width and the W-D slop.

Bibliography

[1] P. M. Kogge and H. S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Trans. Computers*, Vol. C-22, 1973, pp. 786-793.

[2] R.E. Ladner and M.J. Fischer, "Parallel Prefix Computation," *JACM*, Vol. 27-4, 1980, pp. 831-838.

[3] R. P. Brent and H. T. Kung, "A Regular Layout for Parallel Adders," *IEEE Trans. Computers*, Vol. C-31, 1982, pp. 260-264.

[4] T. Han and D. A. Carlson, "Fast Area-Efficient VLSI Adders," *Proc. 8th IEEE Symposium on Computer Arithmetic*, 1987, pp. 49-56.

[5] S. Knowles, "A Family of Adders," *Proc. 15th IEEE Symposium on Computer Arithmetic*, 2001, pp 277-281.

[6] Y. Choi and E. E. Swartzlander, Jr., "Design of a Hybrid Prefix Adder for Non-Uniform Input Arrival Times," *Proc. SPIE 2002 Advanced Signal Processing Algorithms, Architectures, and Implementations XII*, Seattle, 2002.

[7] T. Lynch and E.E. Swartzlander, Jr., "A Spanning Tree Carry Lookahead Adder," *IEEE Trans. Computers*, vol. 41, 1992, pp. 931-939.

[8] I. E. Sutherland and R. F. Sproull, "Logical Effort: Designing for Speed on the Back of an Envelope," in *Advanced Research in VLSI*, Santa Cruz, CA: Univ. of Calif., 1991.

[9] D. Harris and I. Sutherland, "Logical Effort of Carry Propagate Adders," *Proc. 37th Asilomar Conf. Signals, Systems, and Computers*, 2003, pp. 873-878.

[10] H. Dao and V. G. Oklobdzija, "Application of Logical Effort on Delay Analysis of 64-bit Static Carry-Lookahead Adder," *Proc. 35th Asilomar Conf. Signals, Systems, and Computers*, 2001, pp. 1322-1324.

[11] H. Dao and V. G. Oklobdzija, "Application of logical effort techniques for speed optimization and analysis of representative adders," *Proc. 35th Asilomar Conf. Signals, Systems, and Computers*, 2001, pp. 1666-1669.

[12] X. Yu and V. G. Oklobdzija, "Application of Logical Effort on Design of Arithmetic Blocks," *Proc. 35th Asilomar Conf. Signals, Systems, and Computers*, 2001, pp. 872-874.

[13] M. M. Ziegler and M. R. Stan, "A Unified Design Space for Regular Parallel Prefix Adders," *Proc. Design, Automation and Test in Europe Conference and Exhibition*, 2004, pp. 1386-1387.

[14] Z. Huang and M. Ercegovac, "Effect of Wire Delay on the Design of Prefix Adders in Deep-Submicron Technology," *Proc. Design, Automation and Test in Europe Conference and Exhibition*, 2004, pp. 1386-1387.

[15] K. Venkat, "Generalized Delay Optimization of Resistive Interconnections through an Extension of Logical Effort," *Proc. IEEE International Symposium on Circuit and Systems*, 1993, pp. 2106-2109.

[16] L. Dadda, "Some Schemes for Parallel Multipliers," *Alta Frequenza*, Vol. 34, 1965, pp. 349-356.

[17] C. S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Trans. Electronic Computers*, Vol. 13, 1964, pp. 14-17.

[18] V. G. Oklobdzija, D. Villeger, and S. S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach," *IEEE Trans. Computers*, Vol. 45, 1996, pp. 294-306.

[19] V. G. Oklobdzija, "Design and Analysis of Fast Carry-Propagate Adder Under Non-Equal Input Signal Arrival Profile," *Proc. 28th Asilomar Conf. Signals, Systems, and Computers*, 1994, pp. 1398-1401.

[20] B. Parhami, *Computer Arithmetic Algorithms and Hardware Design*, New York: Oxford University Press, 2000.

[21] E.E. Swartzlnader, Jr., *Computer Arithmetic*, Vols. 1 and 2, Los Alamitos, CA: IEEE CS Press, 1990.

[22] Y. Taur and T. H. Ning, *Fundamentals of Modern VLSI Devices*, New York: Cambridge University Press, 1998.

[23] TSMC, *Spice Model of 0.18u LO EPI Process*, 2004.

Vita

Youngmoon Choi was born in Yong-Am, Jun-Nam, Korea on 27 January 1970, the son of Bong-Su Choi and Jung-Sun Kim. He received his B.S. and M.S. degrees in Electrical Engineering from Seoul National University in 1992 and 1994, respectively. From 1994 to 2000, he was with Hyundai Space & Aircraft, Korea, where he worked on the development of the 1'st KOMPSAT satellite. In 2000, he received National Foreign Study Fellowship from the Korean Government and he entered the Graduate School of The University of Texas at Austin. He is a member of Application Specific Processing Group, The University of Texas at Austin, where he focuses his research on computer arithmetic and VLSI circuit design. He worked as an intern at OASIS SiliconSystems in 2002.

Permanent address: 1650 W. 6th St. Apt. Q
Austin, Texas 78703

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.